034115

PHOSPHORUS

Lambda User Controlled Infrastructure for European Research

Integrated Project

Strategic objective:
Research Networking Testbeds

Deliverable reference number: D.2.11

Consolidated Grid-GMPLS Control Plane prototype

Due date of deliverable: 2009-06-30
Actual submission date: 2009-06-30
Document code: Phosphorus-WP2-D2.11

Start date of project:
October 1, 2006

Duration:
33 Months

Organisation name of lead contractor for this deliverable: **Nextworks (NXW)**

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	✓
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Abstract

This document contains the final release notes and operations guidelines for the G^{2}MPLS Network Control Plane prototype.
The G^{2}MPLS prototype consists of all the software modules designed, implemented and publicly demonstrated by the Phosphorus WP2. This deliverable is the result of refinements and fixings to the previously released G^{2}MPLS prototypes (D2.5, D2.10) and contains some further functionalities developed in the final part of WP2, such as the G^{2}MPLS Control Plane resiliency mechanisms, a few new TNRC-SP plugins for L2 Ethernet switches, some tools for the visualization and easy testing of the G^{2}MPLS Control Plane operations.

This deliverable constitutes the last official and public release of G^{2}MPLS Control Plane from the Phosphorus WP2, in the form of a prototypes, open source code parts and operation guidelines.

List of Contributors/Software Developers

Giacomo Bernini	NXW
Gino Carrozzo	NXW
Nicola Ciulli	NXW
Giodi Giorgi	NXW
Francesco Salvestrini	NXW
Giada Landi	NXW
Jakub Gutkowski	PSNC
Adam Kaliszan	PSNC
Lukasz Lopatowski	PSNC
Damian Parniewicz	PSNC
Artur Juszczyk	PSNC
Eduard Escalona	UESSEX

Table of Contents

List of Figures

0 Executive Summary

This document provides the release notes and operations guidelines for the consolidated G^{2}MPLS Network Control Plane. The G^{2}MPLS prototype is delivered in the form of a XEN virtual machine, some parts of which are open source code and therefore released according to the GPLv2 and LGPL v2.1 licences.

In section 3 the contents and basics for operations of the G^{2}MPLS prototypes are described for the different deployment cases: G^{2}MPLS core controller, G^{2}MPLS border controller (i.e. with G.E-NNI), G^{2}MPLS edge controller (i.e. with G.UNI-N) and G^{2}MPLS G.UNI client node (i.e. with guni-gw functionality towards the Grid middleware).

In section 4 the open source code package is described, the additional packages requirements are listed and the G^{2}MPLS stack installation from the source code is explained.

In section 5 the configuration of each module is explained in order to set a general reference for any users in his specific deployment scenario.

In section 6 the general operations guidelines are presented, to ease the step-by-step take-up of the G^{2}MPLS Control Plane.

In Section 7 some examples of G^{2}MPLS deployments are reported, as derived from the multi-domain heterogeneous Phosphorus test-bed.

Appendix A and B report some further examples and a listing of the main configuration commands of the G^{2}MPLS stack.

1 Objectives and Scope

This document briefly describes the contents of the G^{2}MPLS software prototype package. The detailed architectural background and system design notes have been provided in the other WP2 deliverables, listed below:

- D2.1 “The Grid-GMPLS Control Plane architecture”,
- D2.2 “Routing and Signalling Extensions for the Grid-GMPLS Control Plane”
- D2.7 ”Grid-GMPLS network interfaces specification”
- D2.3 “Grid-GMPLS high level system design”
- D2.6 “Deployment models and solutions of the Grid-GMPLS Control Plane”
- D2.8 “Design of the Grid-GMPLS Control Plane to support the Phosphorus Grid AAI”
- D2.9 “Design of Grid-GMPLS interworking with NRPS”

Basic configuration hints and bootstrap procedures are the core of this document, in order to provide a general reference for any users willing to deploy G^{2}MPLS in its own Transport Network with a specific SCN, addressing spaces and interconnections between equipments. Detailed explanations on the available configuration commands exposed by each software module can be retrieved during operation of the stack, by logging into the VTY interface and using the module help.

Most of the G^{2}MPLS modules released in this prototype are based on the QUAGGA routing suite, and therefore the QUAGGA official documentation [QUAGGA-DOC] complements these notes.

A further objective of this document is to produce operation guidelines on G^{2}MPLS to ease its take up by researchers external to WP2 and the Phosphorus project. These guidelines follow a step-by-step approach, starting from the Control Plane preparation down to the configuration of the single G^{2}MPLS modules. Examples of G^{2}MPLS deployments in the Phosphorus test-bed complement and complete the guidelines with live use cases.

2 Terminology

No specific terminology is introduced by this document, which refers to Deliverable D2.1, D2.2, D2.6, D2.7, D2.4, D2.8 and D2.9 for any specific terms used.

A full list of the abbreviations used in this document is provided in Section 9.

3 G²MPLS prototype description

3.1 Package format

The final G²MPLS prototype is released in the form of a XEN virtual machine, configured with all the system packages (libraries and programs) needed for the correct operations of the G²MPLS software modules.

The G²MPLS virtual machine is a XEN Domain U (DomU) based on a Linux/Gentoo distribution for x86 32-bit platforms. It is built with XEN capabilities activated in its kernel 2.6.16. The hosting server from which it has been derived (XEN Domain 0 – Dom0) is a Linux/Ubuntu 7.04 with kernel 2.6.19 and XEN 3.0 installed.

The G²MPLS XEN VM consists of two disks

- g2mpls_controller.sda1, containing the system root ("/")
- g2mpls_controller_swap.sda2, representing the swap memory for that system

The XEN dom0 configuration which is needed to start the G²MPLS XEN VM is provided in the following excerpt to be added as independent file in `/etc/xen/seeds` directory.

```
kernel = "/mnt/xen/vmlinuz-2.6.16-xenU"
memory = 128
name = " g2mpls_controller "
disk = ['file:/mnt/xen/seeds/ g2mpls_controller.sda1,sda1,w', 'file:/mnt/xen/seeds/
g2mpls_controller_swap.sda2,sda2,w']
root = "/dev/sda1 ro"
vif = ['']
cpus = "0-1"
vcpus = 2
```

Code 3-1: G²MPLS XEN VM configuration file in dom0.

The G²MPLS VM boots with pre-configured hostname (`g2mpls-controller`) and IP address for its virtual network interface. Both can be overridden in `/etc/conf.d` according to the user' needs. The root user can be accessed with password "g2mpls".

3.2 Package contents

The G^{2}MPLS prototype comes up with a preconfigured user (user01, password user01) and the main object codes of G^{2}MPLS software modules, as listed in Figure 3-1. These components are contained in the directory `phosphorus-g2mpls` located in `/home/user01`. In details:

- `phosphorus-g2mpls/build` contains the protocols executables and the common libraries of the G^{2}MPLS stack; specifically
  - `phosphorus-g2mpls/build/sbin` groups most of the G^{2}MPLS protocols executables ([PH-WP2-D2.3], [PH-WP2-D2.9]);
  - `phosphorus-g2mpls/build/pyg2mpls` groups all the python components (NCC, CCC and framework tools) just described in [PH-WP2-D2.3];
  - `phosphorus-g2mpls/build/etc` contains the controllers run-scripts and some minimal example of configurations (more detailed configurations are provided in G^{2}MPLS-DEMOS folder);
  - `phosphorus-g2mpls/build/pepgw` groups all the PEPGW components and the GAAATK released by WP4 for AAA integration [PH-WP2-D2.8]
- `phosphorus-g2mpls/G2MPLS-DEMOS` contains all the configurations of the G^{2}MPLS controllers used during the G^{2}MPLS demonstrations in SC'08, ICT'08 and TNC'09 conferences and exhibitions (ref. section 7 for a topology description).

The G^{2}MPLS prototype is based on the Quagga v0.99.7 substrate [QUAGGA-DOC] from which it inherits the base OSPFv2 implementation and some common libraries and tools. Many other functionalities and protocols are implemented in the form of independent processes, also based on the QUAGGA framework. Therefore, most of the G^{2}MPLS modules/processes expose a VTY interface for the inspection and configuration and it is similar to the command line interfaces of the other QUAGGA protocols.

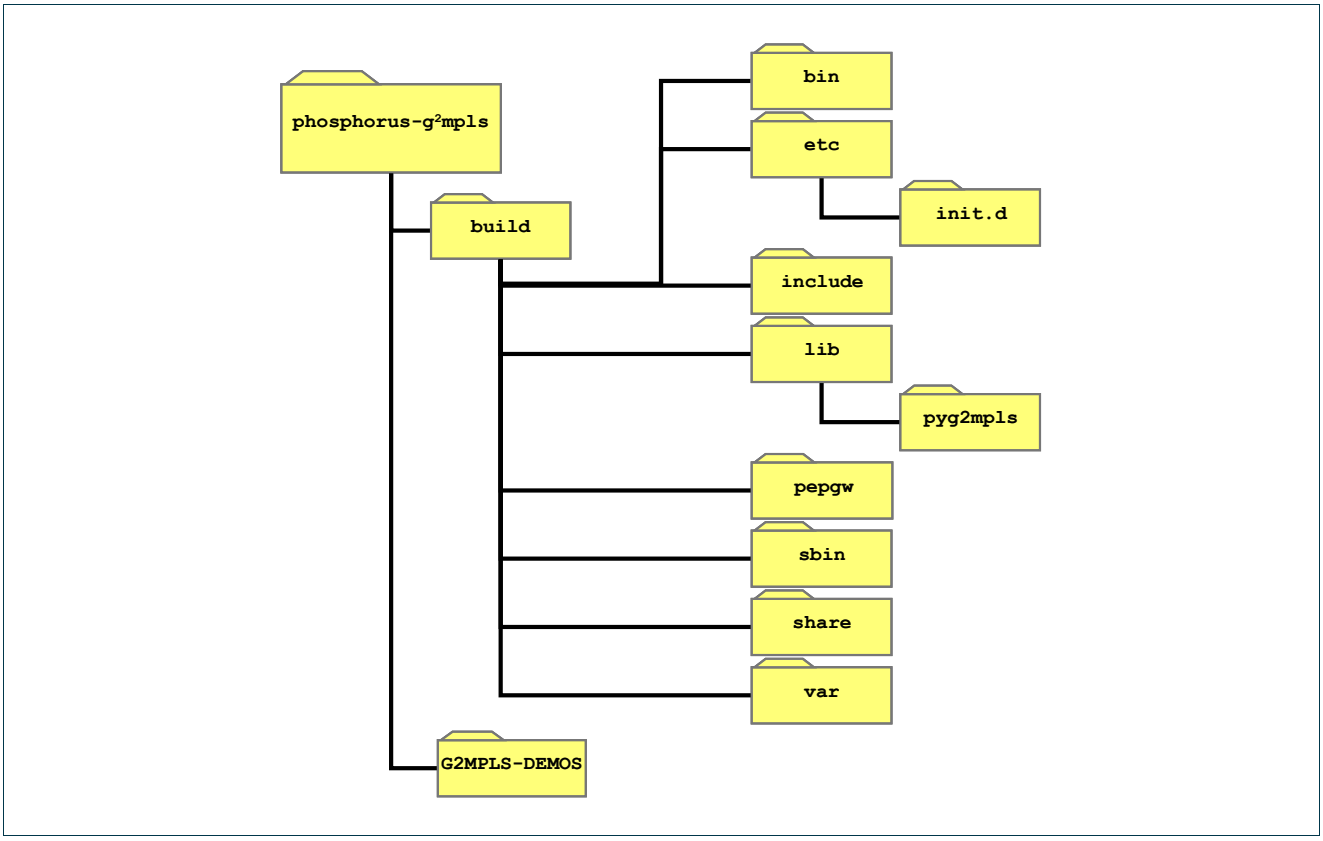

Figure 3-1: Phosphorus G^{2}MPLS prototype build structure.

3.3 Start-up and shut-down procedures

3.3.1 Single protocols

NOTE. This procedure is deprecated for a full controller operation, since most of the protocols depend on the existence of other modules and open CORBA interfaces. These dependencies are preserved by the init scripts for the full G^{2}MPLS controllers (core, border, edge, uni-client) described in sections

Each process in `./build/sbin` can be run with a set of options, briefly described below.

```
"Usage : PROGRAM-NAME [OPTION...]"
                    "Daemon which manages PROGRAM-NAME module"

                    "-d, --daemon       Runs in daemon mode"
                    "-f, --config_file  Set configuration file name"
                    "-i, --pid_file     Set process identifier file name"
                    "-C, --dryrun       Check configuration and exit"
                    "-o, --iors_dir     Set IORs directory"
```

```
                    "-P, --vty_port     Set vty's port number"
                    "-u, --user         User to run as"
                    "-g, --group        Group to run as"
                    "-v, --version      Print program version"
                    "-h, --help         Display this help and exit"
```

Code 3-2: G²MPLS process run options.

Automated bash scripts have been provided in `build/etc/init.d` to start-up, shut-down and restart singularly most of these executables. They locate the configuration files, set some options and run/kill the G²MPLS modules.

Each G²MPLS daemon can be checked for its correct operation and possibly further configured through its VTY, which is a command line interface accessed via telnet to the g2mpls-controller at the ports specified below.

```
TNRCD_VTY_PORT              2613
LRMD_VTY_PORT               2610
SCNGWSD_VTY_PORT            2620
OSPF_VTY_PORT               2604
G2RSVPTED_VTY_PORT          2630
GENNI_G2RSVPTED_VTY_PORT    2631
GUNIN_G2RSVPTED_VTY_PORT    2632
GUNIC_G2RSVPTED_VTY_PORT    2633
G2PCERAD_VTY_PORT           2615
GUNIGWD_VTY_PORT            2614
HG2GWD_VTY_PORT             2625
NCCD_VTY_PORT               2616
```

Code 3-3: G²MPLS main VTY ports.

The configuration file specified as a run option of each daemon contains the VTY commands that are read at the bootstrap of the protocol. Therefore, a set of the available commands per protocol can be inferred by these files or retrieved exhaustively through the help of the VTY interface.

3.3.2 Python modules

NOTE. This procedure is deprecated for a full controller operation, since most of the protocols depend on the existence of other modules and open CORBA interfaces. These dependencies are preserved by the init scripts for the full G²MPLS controllers (core, border, edge, uni-client) described in sections

The binaries for NCC, CCC, RC and PC modules are located in `./build/lib/pyg2mpls`, which is linked by the python site-packages location on the VM (`/usr/lib/python.x.x/site-packages/`). The start-up/shut-down procedures for these modules are wrapped by a bash script:

```
`build/lib/tools/pyrun.sh build/lib/tools/pyg2` <action> <protocol>
      <action>      := { start | stop | restart }
      <protocol>    := { nccd | cccd | rcd | pcd }
```

```
      <instance_no> := number of protocol instances to be run on the same controller
                       (just for the single-module debugging purposes)
```

Code 3-4: G^{2}MPLS Python objects run script.

NOTE. the execution of any xCC modules, of RCD and of PCD generates persistency files (*.pdb and *RSVP DB) that must be removed for a correct restart from scratch of the modules.

All the activities of these python modules depend on actions from G^{2}MPLS protocols or CORBA interfaces. Therefore, the VTY interface they export is a stand-alone process, the nccd, and is mainly used for show commands and creation of Soft Permanent Connections (SPC).

3.3.3 G^{2}MPLS Controllers

NOTE. In order to simplify all the operations, it is suggested to run the controllers described below from the root user. This is necessary for those G^{2}MPLS protocols using the system sockets.

Reference init bundle scripts are

```
build/etc/init.d/g2mpls-core-ctrl

build/etc/init.d/g2mpls-border-ctrl

build/etc/init.d/g2mpls-edge-ctrl

build/etc/init.d/g2mpls-uniclient-ctrl
```

3.3.3.1 *G^{2}MPLS core controller*

The modules needed by a G^{2}MPLS core controller are:

- tnrcd, i.e. the process in charge of implementing the mediation between Control Plane and the Transport Plane equipment;
- lrmd, i.e. the process storing the Control Plane data model and the internal bindings between resources
- scngwsd, i.e. the process that bridges the set of SCN interfaces with the TE-Links and related Control Channels;
- the python pcd;
- g2rsvpted, i.e. the process implementing the G.I-NNI G2.RSVP-TE;
- ospfd, i.e. the process implementing the I-NNI G2.OSPF-TE;
- g2pcerad, i.e. the process implementing the routing algorithms on the G^{2}MPLS multi-domain topologies.

A G^{2}MPLS core controller can be run with the init script `build/etc/init.d/g2mpls-core-ctrl` in which the correct launch sequence is preserved across different start/stop/restart events.

NOTE. In case of failure of any G^{2}MPLS daemons a full restart of the controller must be done, due to the lack of process dependency tracking in this preliminary release.

3.3.3.2 *G^{2}MPLS border controller*

The modules needed by a G^{2}MPLS border controller are:

- tnrcd;
- lrmd;
- scngwsd;
- pepgw
- the python nccd, rcd and pcd;
- g2rsvpted implementing the G.I-NNI G2.RSVP-TE;
- g2rsvpted implementing the G.E-NNI G2.RSVP-TE;
- ospfd, i.e. the process implementing the I-NNI G2.OSPF-TE;
- g2pcerad, i.e. the process implementing the routing algorithms on the G^{2}MPLS multi-domain topologies.
- nccd, i.e. the process implementing the VTY interface to the python xCCs modules

A G^{2}MPLS border controller can be run with the init script `build/etc/init.d/g2mpls-border-ctrl` in which the correct launch sequence is preserved across different start/stop/restart events.

NOTE. In case of failure of any G^{2}MPLS daemons a full restart of the controller must be done, due to the lack of process dependency tracking in this preliminary release.

3.3.3.3 *G^{2}MPLS edge controller*

The modules needed by a G^{2}MPLS edge controller are:

- tnrcd;
- lrmd;
- scngwsd;
- pepgw
- the python nccd, rcd and pcd;
- g2rsvpted implementing the G.I-NNI G2.RSVP-TE;
- g2rsvpted implementing the G.UNI-N G2.RSVP-TE;
- ospfd, i.e. the process implementing the I-NNI G2.OSPF-TE and the UNI flooding of Grid information;

- g2pcerad, i.e. the process implementing the routing algorithms on the G^{2}MPLS multi-domain topologies.
- nccd, i.e. the process implementing the VTY interface to the python xCCs modules

A G^{2}MPLS edge controller can be run with the init script `build/etc/init.d/g2mpls-edge-ctrl` in which the correct launch sequence is preserved across different start/stop/restart events.

NOTE. In case of failure of any G^{2}MPLS daemons a full restart of the controller must be done, due to the lack of process dependency tracking in this preliminary release.

3.3.3.4 *G^{2}MPLS G.UNI CLIENT controller*

The modules needed by a G^{2}MPLS G.UNI CLIENT controller are:

- tnrcd, always using a simulator for the Transport Plane;
- lrmd;
- scngwsd;
- pepgw
- the python ccd and pcd;
- g2rsvpted implementing the G.UNI-C G2.RSVP-TE;
- ospfd, i.e. the process implementing the UNI flooding of Grid information;
- gunigwd, i.e. the process implementing the gateway functionality between the G^{2}MPLS protocols and the WS-Agreement interface towards the Grid Middleware.
- the python g2dialer, i.e. an additional module implementing a direct bridging between the application layer (in particular DDSS) and the G^{2}MPLS protocols.

A G^{2}MPLS G.UNI CLIENT controller can be run with the init script `build/etc/init.d/g2mpls-uniclient-ctrl` in which the correct launch sequence is preserved across different start/stop/restart events.

NOTE. In case of failure of any G^{2}MPLS daemons a full restart of the controller must be done, due to the lack of process dependency tracking in this preliminary release.

4 G²MPLS installation from source code

4.1 OS package contents

The G²MPLS source code is composed of parts released under open source licences and parts which are proprietary and thus not publicly released.

The G²MPLS restricted code parts include the python modules and the pepgw.

The G²MPLS open source code is contained in the directory `phosphorus-g2mpls/src` located in `/home/user01`. In Figure 4-1 is listed its structure:

- `phosphorus-g2mpls/src/g2mpls_common` contains the source code related to the G²MPLS common library (*libg2mpls*). This library contains G²MPLS utilities: common types and addresses, the related set/get/print utilities and some external interfaces common types translation utilities. The library is linked by all the modules in the phosphorus-g2mpls package, and it is released under the GNU LGPLv2.1 licence.
- `phosphorus-g2mpls/src/g2mpls_modules` contains the G²MPLS protocols, which extend the Quagga v0.99.7 routing suite [QUAGGA-DOC]. The package includes software components developed from scratch, base Quagga protocols extended for Grid and GMPLS support, additional tools for the automatic generation of FSM skeletons. This source code is free software released under GNU GPLv2 licence.
- `phosphorus-g2mpls/src/g2mpls_idl` contains the description files related to the CORBA interfaces *(idl)* between the G²MPLS modules. The interface description files are used by all the G²MPLS modules, including the python ones which have a proprietary licence and this released as object code in the G²MPLS VM.

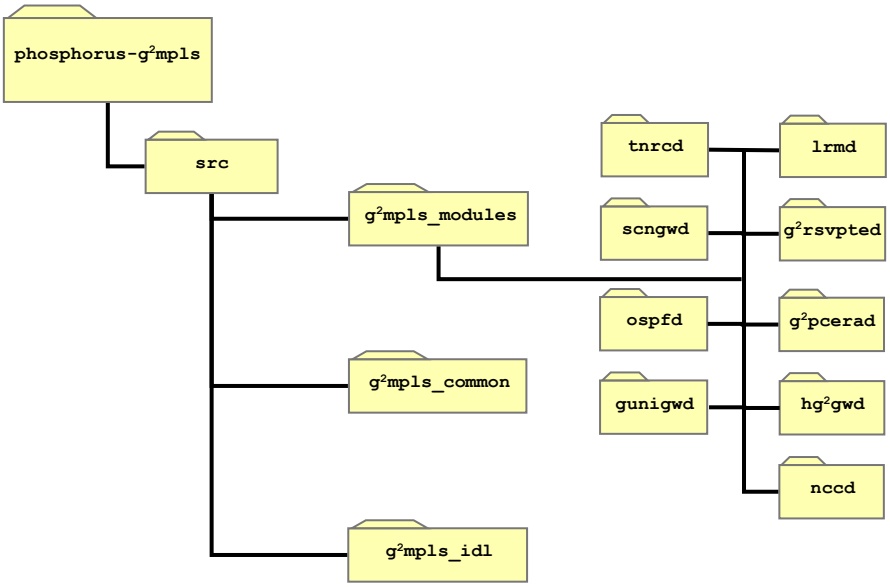

Figure 4-1: Phosphorus G^{2}MPLS source code structure.

4.2 System requirements

The following Linux/Gentoo packages are required to build and use properly the *phosphorus-g2mpls* package:

- *dev-lang/python*: the interpreted, interactive and object-oriented python programming language. It is required to install either version 2.5 or any later 2.x version;
- *dev-python/elementtree*: a light-weight XML object model for python. This package is required to enable the dedicated and proprietary python nccd and cccd signalling, based on XML;
- *net-misc/omniORB*: a robust high performance CORBA 2 ORB for the inter-process communication. It is required to install either version 4.1.2 or any later version;
- *dev-python/omniorbpy*: a robust high performance CORBA ORB for the python inter-process communication. It is required to install either version 3.0 or any later version;
- *dev-java/sun-jre-bin*: the Sun's Java Runtime Environment, required to allow the interfacing between the python nccd and the Grid-AAA layer through the pepgw.

4.3 Installation procedure

The phosphorus-g2mpls source code installation is based on the GNU autotools suite. The autotools suite is mainly composed by three different GNU tools: autoconf (http://www.gnu.org/software/autoconf), automake (http://www.gnu.org/software/automake) and libtool (http://www.gnu.org/software/libtool).

The package comes with a set of scripts built during the development process. The more important scripts are the "*configure*", that are available in the *phosphorus-g2mpls/src/g2mpls_common/* and *phosphorus-g2mpls/src/g2mpls_modules/* directories. These two scripts attempt to guess correct values for various system-dependent variables to be used during compilation, and should be run in order to prepare the source tree to be built on a particular system.

A user who wants to compile and install the phosphorus-g2mpls package in a single shot, avoiding dependencies issues related to the G^{2}MPLS common library (needed and linked by all the modules in the *phosphorus-g2mpls/src/g2mpls_modules/*) and not caring of all the possible "*configure*" options, should run a wrapper script available in the root source code directory: "*build.sh*". This script runs the "*configure*" scripts in each subdirectory with the proper options, builds and installs the G^{2}MPLS common library and all the G^{2}MPLS modules. The actual build and installation process is performed using the gcc/g++ make program.

```
/home/user01/phosphorus-g2mpls/src $./build.sh
Usage: build.sh [COMMAND]

Commands:
       bc | build-common
       bm | build-modules
       ba | build-all
       cc | clean-common
       cm | clean-modules
       ca | clean-all
```

The destination directory for G^{2}MPLS executables and libraries is preconfigured by the build.sh script to be *phosphorus-g2mpls/build*. The usual command that should be invoked from the root phosphorus-g2mpls directory to build the source code and install all the executables, object files and common libraries in the *phosphorus-g2mpls/build* directory is:

```
./build.sh build-all
```

On the other hand, to remove all the program binaries, object files and files created by "*configure*" (allowing the user to build the package for a different kind of system) the command is:

```
./build.sh clean-all
```

A user who wants to compile and install the phosphorus-g2mpls package manually, building separately the G^{2}MPLS common library and the G^{2}MPLS modules, should run the "*configure*" scripts in the *phosphorus-g2mpls/src/g2mpls_common/* and *phosphorus-g2mpls/src/g2mpls_modules/* directories with proper options. Regarding the G^{2}MPLS common library, the minimal set of options needed by the "*configure*" script is:

```
phosphorus-g2mpls/src/g2mpls_common $./configure     --prefix=PREFIX     \
                                                     --localstatedir=DIR \
                                                     --with-idl-path=ARG
```

where:

- *--prefix=PREFIX*: install architecture-independent files in *PREFIX* (default */usr/local*, recommended *phosphorus-g2mpls/build*);
- *--localstatedir=DIR*: put modifiable single machine data in *DIR* (recommended *phosphorus-g2mpls/build/var*);
- *--with-idl-path=ARG*: specify with *ARG* the location of the IDL files for the inter-process communication description (recommended *phosphorus-g2mpls/src/g2mpls_idl*).

To list all the G^{2}MPLS common library "*configure*" options:

```
phosphorus-g2mpls/src/g2mpls_common $./configure  --help
```

After the "*configure*" script has been run successfully, to compile the G^{2}MPLS common library source code and install the library and object files in the specified directory the user must run:

```
phosphorus-g2mpls/src/g2mpls_common $make && make install
```

Regarding the G^{2}MPLS modules, there are a lot of options to be set in the "*configure*" script: many Quagga protocols must be disabled because not needed in the G^{2}MPLS stack. In particular the user must run the "*configure*" script with at least these options:

```
phosphorus-g2mpls/src/g2mpls_modules $./configure    --disable-ipv6            \
                                                     --disable-bgpd            \
                                                     --disable-bgp-announce    \
                                                     --disable-ripd            \
                                                     --disable-ripngd          \
                                                     --disable-ospf6d          \
                                                     --disable-rtadv           \
                                                     --disable-capabilities    \
                                                     --disable-xxxd            \
                                                     --disable-snmp            \
                                                     --disable-watchquagga     \
                                                     --enable-tcp-zebra        \
                                                     --enable-vtysh            \
                                                     --enable-ospf-te          \
```

```
                                                --enable-opaque-lsa       \
                                                --prefix=PREFIX           \
                                                --localstatedir=DIR       \
                                                --with-idl-path=ARG       \
                                                --with-libg2mpls-path=ARG \
                                                --with-run-user=ARG       \
                                                --with-run-group=ARG
```

where:

- *--disable-ipv6*: turn off Quagga IPv6 related features and protocols;
- *--disable-bgpd*: do not build bgpd directory for BGP protocol;
- *--disable-bgp-announce*: turn off BGP route announce;
- *--disable-ripd*: do not build ripd directory for RIP protocol;
- *--disable-ripngd*: do not build ripngd directory for RIP-ng protocol;
- *--disable-ospf6d*: do not build ospf6d directory for OSPFv6 protocol;
- *--disable-rtadv*: disable IPv6 router advertisement feature;
- *--disable-capabilities*: disable using POSIX capabilities;
- *--disable-xxxd*: do not build xxxd directory for XXX test daemon;
- *--disable-snmp*: disable SNMP support;
- *--disable-watchquagga*: do not build watchquagga monitoring suite;
- *--enable-tcp-zebra*: enable TCP/IP socket connection beetwen zebra and other daemons;
- *--enable-vtysh*: include integrated vty shell for Quagga;
- *--enable-ospf-te*: enable Traffic Engineering Extensions to OSPF;
- *--enable-opaque-lsa*: enable OSPF Opaque-LSA with OSPFAPI support (RFC2370);
- *--prefix=PREFIX*: install architecture-independent files in *PREFIX* (**must be the same as for G^{2}MPLS common library**);
- *--localstatedir=DIR*: put modifiable single machine data in *DIR* (**must be the same as for G^{2}MPLS common library**);
- *--with-idl-path=ARG*: specify with *ARG* the location of the IDL files for the inter-process communication description (**must be the same as for G^{2}MPLS common library**);
- *--with-libg2mpls-path=ARG*: specify with *ARG* the location of the G^{2}MPLS common library directory (recommended *phosphorus-g2mpls/src/g2mpls_common*);
- *--with-run-user=ARG*: specify with *ARG* the user to run G^{2}MPLS modules as (default “quagga”);
- *--with-run-group=ARG*: specify with *ARG* the group to run G^{2}MPLS modules as (default “quagga”).

After the “*configure*” script has been run successfully, to compile the G^{2}MPLS modules source code and install the library and object files in the specified directory the user must run:

```
phosphorus-g2mpls/src/g2mpls_modules $make && make install
```

5 G²MPLS prototype configuration

The configuration files distributed with the G²MPLS prototype represent a simple and not exhaustive reference, released just to let the controller boot and start its operation. A user should customize those configurations and possibly extend them in order to fit his choices about address spaces, functionalities and Control Plane scenarios to implement.

Each G²MPLS controller (edge, core, border, client) uses a particular set of modules. It is crucial that all running protocols have appropriate configuration files placed in *build/etc/* directory.

Each configuration file contains the VTY commands that are read at the bootstrap of the specific module (a list of all the available commands is in Appendix A). A few common commands contained in all configuration files are used to set the password required when accessing VTY via telnet and to specify the log file.

During the building and compilation process, some sample configuration files are created in *build/etc*, containing minimal G²MPLS controller configuration. However, to run a G²MPLS controller, the sample configuration files must be renamed (i.e. the *".sample"* removed from the configuration file name). More detailed configurations that can be used as examples are available in */home/user01/phosphorus-g2mpls /G²MPLS-DEMOS* folder of G²MPLS prototype XEN image.

Most of the Control Plane configurations are contained in the configuration files of tnrcd and lrmd. The lrmd module acts as a hub element for all the protocols, in particular the ospfd and the g2rsvpted.

NOTE. Python modules configuration files are not located in *./build/etc. Please see Section 5.1 for more details.*

5.1 Python modules configuration

Each python module (NCC, CCC, RC, PC and G^{2}dialer) has its own configuration file, *config.py*. There is also a root configuration file, named *rootconfig.py*, containing common settings and configurations for all the python modules.

The *rootconfig.py* file specifies:

- the directory for the **.ior* CORBA client and servants files (should be */home/user01/phosphorus-g2mpls/build/var*);
- the directory for all the modules log files (should be */home/user01/phosphorus-g2mpls/build/var*);
- the location of the AAA PEP;
- log levels for common log info to all python modules (e.g., timers, CORBA, network activities);
- LRM CORBA client description (e.g., **.ior* file name, module and interface name).

Each *config.py* specifies for the related module:

- module name;
- log levels for module-specific log info (e.g., FSM, timers, CORBA, network activities);
- CORBA clients and servants description (e.g., **.ior* file name, module and interface name);
- network configuration for UDP communication (only for NCC and CCC);
- FSM timers (only for NCC and CCC);
- persistency DB filename (only for NCC, CCC and RC).

5.2 G2.RSVPTE configuration

Depending on G^{2}MPLS controller type different G^2.RSVPTE processes are launched. Therefore, it is important to provide proper configuration files for all processes. Table 5-1 summarizes configuration files names required by particular controllers.

G^{2}MPLS controller	G^2.RSVPTE configuration file
G^{2}MPLS core controller	• *ginni.core.g2rsvpted.conf*
G^{2}MPLS border controller	• *genni.g2rsvpted.conf* • *ginni.border.g2rsvpted.conf*
G^{2}MPLS edge controller	• *guni.g2rsvpted.conf* • *ginni.border.g2rsvpted.conf*

G^{2}MPLS client controller	• *gunic.g2rsvpted.conf*

Table 5-1: G^2.RSVPTE configuration files required by different types of G^{2}MPLS controllers

NOTE. The sample configuration files doesn't need to be edited, only renamed.

5.3 G2.OSPFTE configuration

The configuration file for G2.OSPF-TE is *ospfd.conf*. This configuration is used to define which G2.OSPFTE (I-NNI, E-NNI, UNI-N, UNI-C) instances should be created, depending on the G^{2}MPLS controller type. Additional commands may be used to set suitable logging level.

5.4 G2.PCERA configuration

The Path Computation Engine Routing Algorithm module requires configuration file *g2pcerad.conf*.

NOTE. The sample configuration files doesn't need to be edited, only renamed.

5.5 LRM configuration

The Control Plane resources are all maintained by the LRM. In lrmd.conf the Control Plane logical topology is detailed in terms of:

- router ID of the G^{2}MPLS controller
- SCN interfaces used to receive/transmit protocol packets
- Control Channels
- TE-links with their TE attributes (adjacency type, TE metric, colours, SRLGs, TNAs, etc.)
- Data-links (in 1:1 correspondence with those loaded by tnrcd and exported at the TNRC's CORBA interface)
- bindings of TE-links with Control Channels
- insertion of Data-links into TE-links.

This information is centralized and used by all the protocols for routing and signalling. Therefore, the lrmd configuration file is larger than the configuration files of the upper protocols, which inherit most of the information from it.

Example of LRM configuration file is available in Appendix B.

5.6 TNRC configuration

The Transport Plane data model is stored in tnrcd. *The tnrcd.conf* file directory specifies the location of the equipment. This equipment could be a real Transport Plane equipment or a simulator. In both cases the user must specify the resources (ports and labels) to be used by G^{2}MPLS CP. The simulator option can be very useful to run the G^{2}MPLS Control Plane without connecting to real hardware, a feature that can help the training/learning or the pre-production phases.

The *tnrcd.conf* file also specifies the name and location of equipment specific configuration file. This configuration file provides information about equipment type and its basic characteristics as well as available ports numbers and their properties. Naming convention for these files is presented in Table 5-2. The structure of all equipment configuration files used by tnrcd is the same.

Equipment type/name	Default name of equipment configuration file
Device simulator	*tnrcd.eqpt.sim*
Calient DiamondWave FiberConnect	*tnrcd.eqpt.calient*
ADVA FSP 3000RE-II	*tnrcd.eqpt.adva*
Foundry XMR 8000	*tnrcd.eqpt.foundry*
Foundry MLX-16	*tnrcd.eqpt.foundry*
Allied Telesis AT-8000S	*tnrcd.eqpt.at*
Allied Telesis AT-9424T	*tnrcd.eqpt.at*
Czech Light Switch	*tnrcd.eqpt.cls*

Table 5-2: Equipment specific tnrcd configuration files default names

Example of TNRC equipment simulator configuration files is available in Appendix B. These configuration files are valid also for corresponding real equipment configuration files.

5.7 SCNGW configuration

SCNGW module requires configuration file *scngwsd.conf*.

NOTE. The sample configuration files doesn't need to be edited, only renamed.

5.8 GUNIGW configuration

GUNIGW module requires configuration file *gunigwd.conf*.

NOTE. The sample configuration files doesn't need to be edited, only renamed.

5.9 HG2GW configuration

HG2GW module requires configuration file *hg2gwd.conf*.

NOTE. The sample configuration files doesn't need to be edited, only renamed.

6 G²MPLS Control Plane operations guidelines

This section presents some guidelines for the preparation and installation of the G²MPLS Control Plane. It is composed of step-by-step actions to be done:

a) Preparation of G²MPLS CP configuration details,

b) Installation and configuration of G²MPLS controllers,

c) Connecting Grid applications (optional),

d) Configuring the policy system (optional),

e) Interfacing to Harmony system (optional).

6.1 Control Plane details preparation

The G²MPLS Control Plane is composed of a set of consistently configured G²MPLS controllers. Its configuration is based on already existing Transport Plane thus before starting the G²MPLS preparation, it is very useful to prepare overview of the Transport Plane resources. Using the Transport Plane overview, knowing where Grid users are connected to the network and knowing points of connection to other networks, the G²MPLS CP topology details and SCN connectivity details can be prepared. Having G²MPLS CP schemas with all components details, the installation and configuration process of the G²MPLS controllers can start.

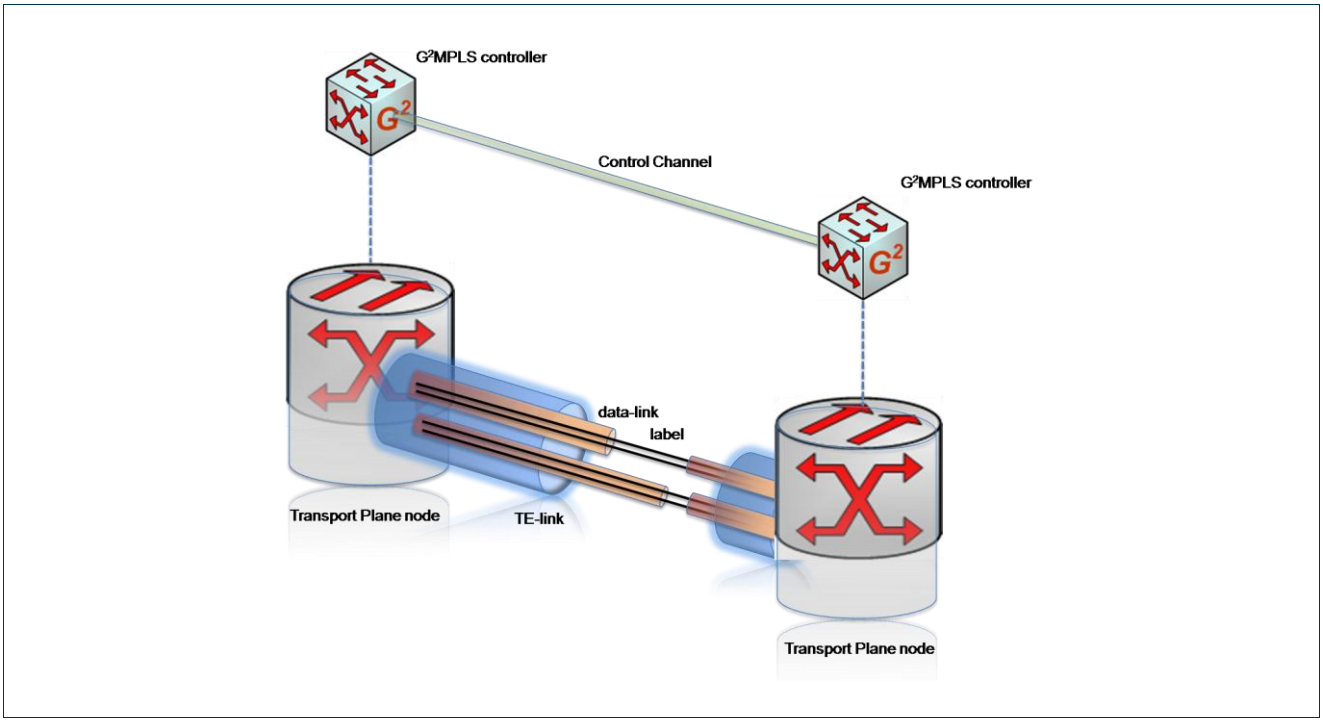

Figure 6-1: G²MPLS Control Plane and Transport Plane description components.

NOTE. There is no need to prepare Grid resource descriptions in order to run G²MPLS Control Plane. The Grid resources are dynamically published to G²MPLS Control Plane by Grid applications.

6.1.1 Transport Plane resources description

In G²MPLS, the Transport Plane resources are modelled as set of nodes, data-links, label resources, TE-links and TNAs. These resources are basic components of Transport Plane network topology description.

Each Transport Plane node is identified using a router-id identifier of its corresponding G²MPLS controller. Each G²MPLS controller can control only one Transport Plane device (or a physical partition of it). The router-id is represented in form of IPv4 address.

Each data-link is identified using both endpoints identifiers. The data-link endpoint identifier is composed of identifiers for the equipment board and port where the data-link is installed. The data-link endpoint identifier has 32 bits representation. Data-links are characterized also by switching type, bandwidth, protection, etc.

In case of DWDM, SONET/SDH or Ethernet VLANs Transport Plane systems, the Transport Plane resource description should contain the label resources representing respectively: a lambda frequency, a timeslot or a VLAN ID. The label resource identifier is represented as 32 or 60 bits value depending on the used switching capability. The label resource must be always bound to a data-link.

In order to achieve better scalability, the data links under the same adjacency and with compatible TE data are bundled in TE-links. Each TE-link is also identified using both endpoints identifiers in form of IPv4, IPv6 or unnumbered addresses. TE-links contain TE attributes: adjacency type, TE metric, colours, SRLGs, TNA, etc.

There are three kinds of TE-link regarding adjacency type:

- I-NNI TE-link (connecting nodes located in the same network domain),
- E-NNI TE-link (connecting nodes located in two different network domains),
- UNI TE-link (connecting user to the network domain).

Additionally, there is a TNA identifier related to each UNI-TE-link, representing user-network attachment point. It is presented in form of IPv4, IPv6 or NSAP addresses.

NOTE. Depending on the type of G^{2}MPLS controller, the TE-links tat can be installed are:
- G^{2}MPLS core controller: only I-NNI TE-links are installed,
- G^{2}MPLS border controller: only I-NNI TE-links and E-NNI TE-links are installed,
- G^{2}MPLS edged controller: only UNI TE-links and I-NNI TE-links are installed,
- G^{2}MPLS client controller: only UNI TE-links are installed.

6.1.2 SCN connectivity requirements

All the G^{2}MPLS controllers communicate with peers through the Signalling Communication Network. The SCN provides the connectivity for signalling and routing protocols. There are following SCN segments:

- I-NNI segment, containing all controllers belonging to a domain,
- UNI segment, containing one G^{2}MPLS edge controller and one G^{2}MPLS client controller,
- E-NNI signalling segment, containing G^{2}MPLS border controllers of two adjacent G^{2}MPLS domains,
- E-NNI routing segment (inter-domain routing), containing all G^{2}MPLS controllers with Routing Controller module instance running.

There should be one SCN IP address configured per each kind of SCN segment in which the G^{2}MPLS controller participates. Private IP addresses space could be used in SCN segments. For stack management purposes, each G^{2}MPLS controller should have a separated management IP address (public or private IP address).

NOTE. The SCN partitioning in segments allows to avoid any routing between the different areas, thus enforcing networks privacy/security issues.

In the SCN segment, G^{2}MPLS Control Channels are configured to provide direct communication between two G^{2}MPLS controllers. The Control Channel must exist for each adjacency available in form of TE-link. Each Control Channel is identified using a 32 bit identifier.

NOTE. There should be no point-to-point Control Channels declared in E-NNI routing segment. E-NNI routing segment communication is broadcasted to every connected Routing Controller.

SCN ENNI segments must be available in two or more administrative domains. There are two main possibilities to create the SCN ENNI segments:

- Any VPN solution over Internet,
- Using additional data-link connection between both domains.

6.2 G²MPLS Control Plane installation

This section presents the installation procedures and some configuration tips when using the G²MPLS XEN virtual machines. These VMs have been prepared by Phosphorus-WP2 development team and are available for download from the Phosphorus website (http://www.ist-phosphorus.eu/deliverables.php).

6.2.1 Host system preparation with XEN virtualization platform

Due to the possibly high number of G²MPLS controllers to be run for each G²MPLS domain, it would be effective the use some virtualization solutions to collapse in a single server platform many controllers. The G²MPLS development team decided to use XEN virtualization platform and all following procedure are related to that platform.

Installing XEN on a hosting machine depends on its installed OS. In Ubuntu Linux, the XEN server package is part of the official repositories and can be easily installed through apt-get:

```
apt-get install ubuntu-xen-server
```

After the host machine reboot the command:

```
sudo xm list
```

should show *Domain-0* which is a virtualization supervisor in the host system.

The basic management of the XEN virtual machines can be done with four simple commands presented in Table 6-1.

Command	Description
sudo xm list	list of all running virtual machines

sudo xm create *vm_starting_file*	start new virtual machine
sudo xm shutdown *domain_name*	kill particular virtual machine
sudo xm console *domain_name*	access to the console of particular virtual machine

Table 6-1: Most useful XEN commands

The required amount of RAM memory in the host running the XEN server depends on the number, type and size of the virtual machines that will be used in the system. The G^{2}MPLS CP prototype virtual machine needs at least 32 MB, with a recommended RAM of 128 MB. The minimum size of the memory left for the host server should be 256 MB. For running a few instances of the G^{2}MPLS CP prototype virtual machines, a RAM memory size of 1GB or greater is suggested on the host server, and a minimum of 2 GB is recommended.

6.2.2 G^{2}MPLS controller configuration

To start using the G^{2}MPLS controller prototype, the configuration files and configuration of network interfaces must be modified.

According to what kind of controller will be configured the appropriate set of configuration files must be prepared. Sections 3.3.3 and 5 describe the configuration files and their location.

The configuration of the network interfaces in each controller must be done in compliance with the SCN and network management schemes. The network interfaces addresses have to be the same as in G^{2}MPLS lrmd configuration file.

NOTE. All SCN IP addresses present in G^{2}MPLS lrmd configuration file must be configured in the network interface(s).

6.3 G^{2}MPLS interfacing with Grid middleware and applications

6.3.1 G^{2}MPLS with DDSS application

The DDSS application is a backup files solution that offers the possibility for the user to store the files in a free distributed disk space of the storage servers. The DDSS application uses grid-ftp as back-end, which is a very efficient transport data protocol for high-bandwidth and big files transmission.

In the Phosphorus WP2-WP3 joint activity, this application was adapted to interface with G^{2}MPLS CP and request a high bandwidth path to a storage server. The storage server can be defined explicitly by the user or chosen by the G^{2}MPLS CP (anycasting) between the most suitable to handle the file backup at that time. Using a dynamically provisioned high bandwidth path, the DDSS application is able to fast backup a user file on the

server. After the file transmission, the path in the G^{2}MPLS network is terminated and all the network resources released.

The DDSS application is interfaced with the G^{2}MPLS CP using clients, provided by WP2 team, of the G2.CCC module Corba interface. The clients are used in two locations:

- In each storage system where it is injecting information about free disk space available in the local storage system to the G2MPLS client,
- In the DDSS GUI application where it is used to pass path setup and teardown requests to G^{2}MPLS client.

The client is available in *src/pyg2mpls/g2dialer*.

6.3.2 G^{2}MPLS with KODAVIS application

The KODAVIS application uses a remote server to combine atmospheric processes and a client to visualize the results in an interactive globe. G^{2}MPLS interfaces with the KODAVIS application by means of the Unicore MSS and G.UNI-GW which is located at the GUNI client nodes. The G.UNI-GW translates any WS-AG requests into the CORBA methods that trigger the G^{2}MPLS Call and LSP setup (client side). Moreover, the G.UNI-GW injects grid site information into the control plane. G.UNI-GW implements the BES Web Service for signalling and the GRR Web Service for routing.

The main routing action involves the injection of Grid capabilities and availability from the application server to the G^{2}MPLS CP. On the other hand, the main signalling actions involve the request of network connections to G.UNI-GW which triggers the G^{2}MPLS Call and LSP setup after contacting the CCC. G.UNI-GW uses two different ports for routing or signalling purposes, thus, the MSS must know the URL or IP address and which port to connect for either BES or GRR services. The communication between G.UNI-GW and the CCC is carried out internally using the CallController (Mgmt) and g2mplsTopology IDLs.

G.UNI-GW is available in *src/gunigwd.*

6.4 G^{2}MPLS interfacing with the Harmony system

The process that enables the interoperation of G^{2}MPLS and Harmony is HG^{2}GW, which implements a reservation Web Service that maps NS requests from one side to the other. HG^{2}GW initially starts the reservation Web Service and waits for incoming requests from Harmony on a configured port. The Web Service is called using standard SOAP/XML messages according to the reservation WSDL. On the other side, HG^{2}GW implements the client/server side of the CORBA interface that communicates with the NCC.

HG^{2}GW should be started at a G^{2}MPLS border node in order to work as an inter-domain gateway between Harmony and G^{2}MPLS. In a communication with direction Harmony-G^{2}MPLS, the reservation Web Service messages are called by Harmony, so Harmony must know the URL or IP address and port in which HG^{2}GW is

located. In a communication G^{2}MPLS-Harmony, the gateway must know the URL and port of the Harmony IDB to which the requests have to be sent. On the CORBA side, HG^{2}GW is in the same machine of the NCC so the communication is carried out internally using the CallController (Mgmt and EW) and g2mplsTopology IDLs.

HG^{2}GW is available in *src/hg2gwd.*

6.5 G^{2}MPLS interfacing with the Grid-AAA layer

The interfacing of the G^{2}MPLS CP with the Grid-AAA layer occurs in three G^{2}MPLS controllers:

- in the ingress domain, the ingress G^{2}MPLS edge controller
  - Network Call Controller (NCC-1)
  - PEP-GW
  - GAAA-TK
- in the transit/egress domain, the ingress G^{2}MPLS border controllers
  - Network Call Controller (NCC-n)
  - PEP-GW
  - GAAA-TK
- in the egress G^{2}MPLS UNI client
  - Client Call Controller (CCC-z)
  - PEP-GW
  - GAAA-TK

PEP-GW is always present in any G^{2}MPLS controller (except for the core ones), to allow a dynamic selection and mediation of the incoming AuthZ/AuthN requests. The actions on the transit/egress domain and the egress client are similar and just token-based. On the contrary the actions on the ingress domain are based on context information regarding the call and tokens. In details,

- NCC-1 requires authorization for the incoming G^{2}MPLS call based on both the user credentials and a complete resource description, including source and destination client nodes (TNAs).
  - PEPGW-1 translates these parameters in a set of data structures as required by the GAAA-TK-1 and triggers the authorization procedure on the PEP (PEP.authorizeAction).
  - If the AuthZ result is positive, PEPGW-1 requests TVS to generate a new Global Reservation Identifier (GRI) for the current call, which is the pilot token (TokenBuilder.getXMLToken), and the generated pilot token is returned to NCC-1 for signalling
  - If the AuthZ result is negative, PEPGW-1 notifies NCC-1 for signalling abortion.
  - In case a pilot token has been generated for NCC-1, each downstream NCC-n and CCC-z processes the received GRI (token) and requires its authorization
- PEPGW-n forwards the token validation request to the GAAA-TK-n (TVS.validateToken).
  - the AuthZ result is forwarded by the PEPGW-n to NCC-n/CCC-z and signalling is completed accordingly.

The operation of the PEP-GW is completely transparent to the user and a minimal configuration needs to be customized in *pepgw/conf/pepgw.conf* and in the python G^{2}MPLS NCC as described in section 5.1.

6.6 G^{2}MPLS operations common issues and troubleshooting

During the usage of the G^{2}MPLS stack some common issues could occur. All the problems can be divided into two general groups:

- problems related to starting of daemons,
- problems related to the Control Plane operation, in particular the signalling of paths

NOTE. Most of the problems are due to mis-configuration of the G^{2}MPLS CP.

There are few mechanism that could help in analysing and solving these problems:

Step 1) At the beginning, listing all running G^{2}MPLS stack daemons using Linux command:

```
ps aux
```

should be used to investigate which daemon didn't start properly. If more than one daemon didn't start, the problem should be looked for in the daemon which not started as the first.

Step 2) Next step is to check if all needed configuration files are enclosed in proper directory, especially those related to not started protocols daemons.

Step 3) Then if problem is still not identified, VTY interface to particular protocol daemons should be useful (VTY ports of all protocol daemons can be found in section 3.3.1). Looking for information available in each VTY daemons, the information presented by VTY daemon should be compared with G^{2}MPLS CP details schemes. If there is a difference it could be a typo error in the configuration file.

Step 4) If all mechanism mentioned above do not work or do not remove all problems next step is to look into the log files. Each daemon should have its own log file in *build/var/* directory. In case of the problem, there should be some error notification present in the text of the log file. The error notifications should help to find the location of the problem.

If the G^{2}MPLS stack have been compiled from the source code and installed manually, other problems could come out. When some problems appeared in this situation, all steps included in Section 4 should be checked.

7 G²MPLS Control Plane deployment examples

This chapter presents some examples of configuration and deployment of the G²MPLS Control Plane. The examples show how the guidelines from Section 6 were applied for setup of G²MPLS Control Plane in real test-bed scenarios by WP2/WP6 team.

7.1 The Phosphorus multi-domain G²MPLS testbed

7.1.1 Overview

Some optical Transport Plane resources were dedicated for the G²MPLS CP prototype testing and demonstration in the PSNC-Pionier network (Poland) and in the UESSEX (UK) network laboratories.

In order to test most of the G²MPLS functionalities, several local testbeds have been installed using FSC, LSC and L2SC (Ethernet) devices, interconnected through GÉANT2 network. Kodavis and DDSS Grid applications complemented the G²MPLS test-bed on the Grid layer.

An inter-domain data-link to I2CAT local testbed completed the G²MPLS test-bed to enable the interfacing with the Harmony system developed by Phosphorus WP1 team.

7.1.2 PSNC local testbed

In PSNC, there are two testbeds installed:

- the Lambda Switching Capability (LSC) testbed, based on an optical DWDM ring with three ADVA FSP 3000RE-II ROADMs. These equipments have been inter-connected with bi-directional optical fibers and the DWDM system configured to hold up to 40 wavelengths.

- the Layer 2 Switching Capability (L2SC) testbed, build on an optical Ethernet routers with two Foundry NetIron XMR 8000. Each equipment have been partitioned into two independent Ethernet-switching sub-nodes. All sub-switches have been inter-connected with bi-directional optical fibers.

The three inter-domain data-links were connected to LSC testbed:

- two data-links towards UESSEX testbed,
- a data-link towards I2CAT testbed.

The Grid application server nodes, containing both Kodavis and DDSS application and Grid middleware installed, were connected using 1GE connection interfaces to the Transport Plane:

- one server node was connected as a network client to the LSC testbed,
- two server nodes were connected to the L2SC (Ethernet) testbed.

Two additional server nodes were available in purpose of running G^{2}MPLS controllers.

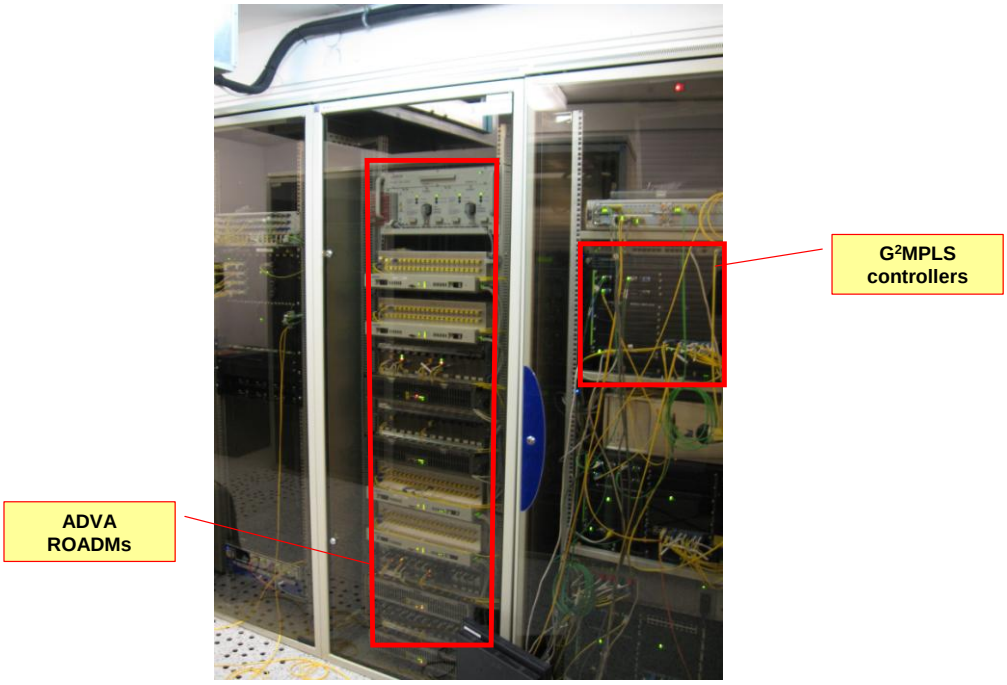

Figure 7-1: Phosphorus G^{2}MPLS testbed in PSNC: LSC part.

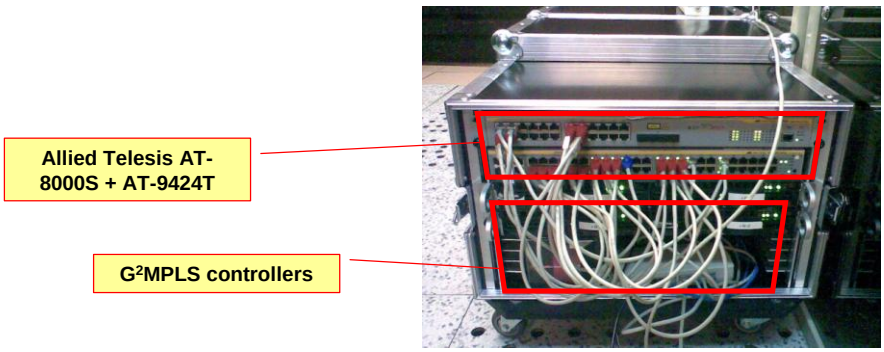

Figure 7-2: Phosphorus G²MPLS testbed in PSNC: L2 Ethernet part.

7.1.3 UESSEX local testbed

In the UESSEX, there is one Fiber Switching Capability (FSC) testbed installed, built around one Calient Diamond Wave Fiber Connect. This equipment is partitioned into four independent fiber-switching sub-nodes. Each sub-switch has been interconnected with bi-directional optical fibers to the other sub-switches to realize a fully meshed topology.

Two inter-domain data-links to PSNC testbed were connected towards FSC testbed.

The three Grid application server nodes, containing both Kodavis and DDSS application and Grid middleware installed, were connected using 1GE connection interfaces to the LSC testbed Transport Plane.

Seven additional servers nodes were available in purpose of running G²MPLS controllers.

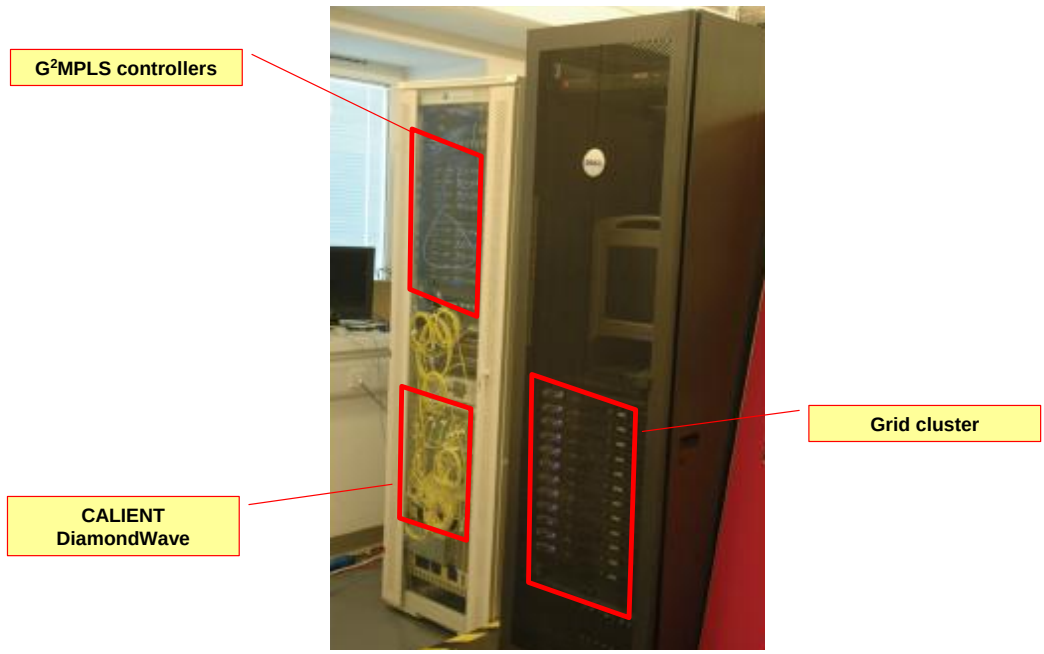

Figure 7-3: Phosphorus G²MPLS testbed in UESSEX: FSC part.

7.1.4 G²MPLS configuration details

This section presents the G²MPLS CP configuration example in case of interconnected PSNC LSC and PSNC FSC local testbeds. Each of the local testbeds has been configured as a standalone G²MPLS domain, thus the Control Plane is composed of two separated domains.

The Transport Plane resources and its interconnections are presented on Figure 7-4. The Transport Plane layout contains TN nodes (LSC ROADM and FSC switch devices), client nodes (application servers) and data-links with TN device port identifiers where the data-links are installed. These three groups of information were mandatory description of Transport Plane which is important for the creation of G²MPLS Control Plane detailed description. Analyzing the Transport Plane topology, there were planned:

- locations of the G²MPLS controllers,
- infrastructure for interconnecting G²MPLS controllers in SCN segments.

The SCN E-NNI segments were created using additional VLAN over GÉANT2 network.

The Transport Plane details of Harmony system is not important for configuration of the G²MPLS CP because of existing of an additional layer of signalling and routing translation between these two different kinds of provisioning systems.

NOTE. The Figure 7-4 contains details about VLANs between PSNC and UESSEX domains but this information is not important for the creation of G^{2}MPLS Control Plane details description. The Transport Plane inter-domain data-links implementation using VLAN technology is not related to any dynamic switching capabilities. These inter-domain data-links are created manually by the network operator.

The SCN details for both LSC PSNC and FSC UESSEX domains are shown on Figure 7-5 where there is specified the following information:

- a kind of G^{2}MPLS controllers stack to be installed (G^{2}MPLS core/edge/client/border),
- an identifier for each of G^{2}MPLS controller,
- location of E-NNI Routing Controller for each domain,
- number of SCN interfaces for each G^{2}MPLS controller,
- G^{2}MPLS controller connections to particular SCN segments,
- SCN IP addresses assigned for each of the connection,
- fully accessible management IP addresses assigned for each of G^{2}MPLS controller.

On the Figure 7-6, it is shown the logical topology of G^{2}MPLS CP based on Transport Plane resources. This particular description contains:

- a kind of G^{2}MPLS controllers stack to be installed (G^{2}MPLS core/edge/client/border),
- an identifier for each of G^{2}MPLS controller,
- I-NNI, UNI and E-NNI TE-links with both endpoints identifiers and TEM parameter,
- an Control Channel identifier for each TE-link,
- client TNA identifiers.

On the Figure 7-6, a virtual “Harmony border controller” is also present, which is not existing in reality but its router id, TE-link endpoint and data-link endpoint identifiers must be inserted in the adjacent G^{2}MPLS border controller configuration.

However, the Figure 7-6 does not present data-links G^{2}MPLS CP components and data-links endpoint identifiers because of lack of free space on the picture.

In PSNC and UESSEX, there were taken two different ways of installation of the G^{2}MPLS controllers. In PSNC, all the four G^{2}MPLS controllers were installed on one server node with XEN solution. In UESSEX, each of the G^{2}MPLS controllers were installed on separated server nodes.

NOTE. A dedicated folder in the released prototype (`/home/user01/phosphorus-g2mpls/G2MPLS-DEMOS`) contains all the configuration files of the G^{2}MPLS modules run on each node. The purpose of this directory is to provide a reference for G^{2}MPLS users about the commands needed to setup the G^{2}MPLS Control Plane in this complex multi-domain scenario.

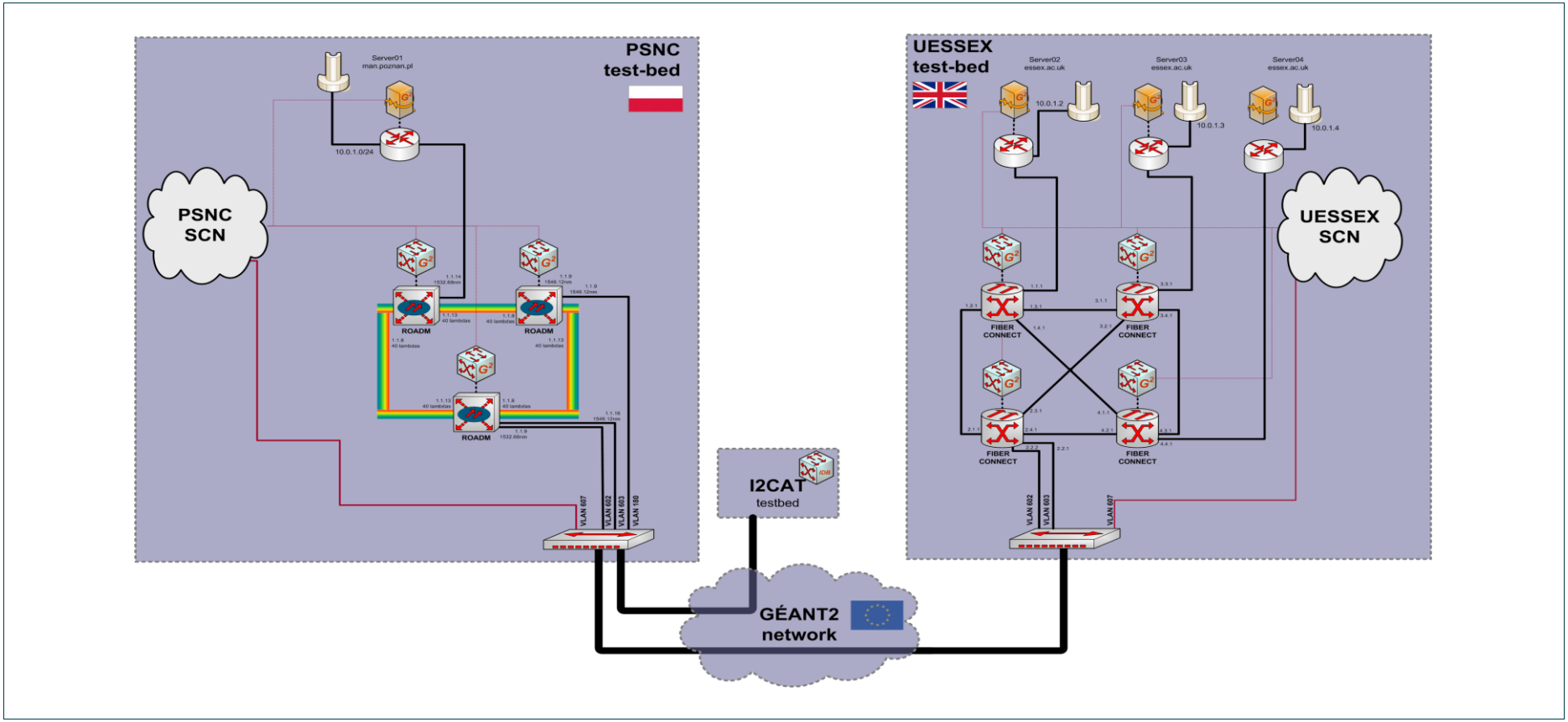

Figure 7-4: Phosphorus G^{2}MPLS testbed: Transport Plane layout.

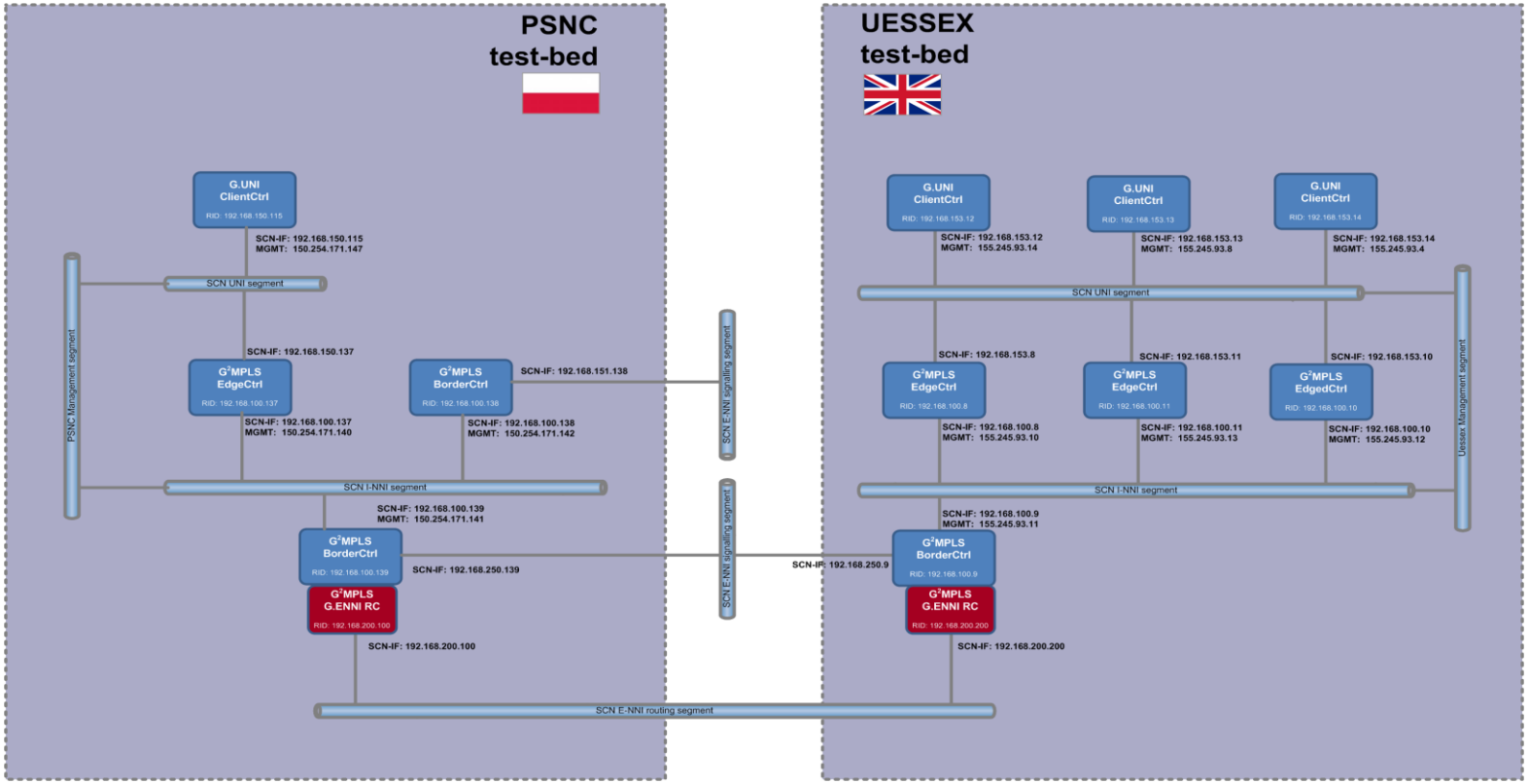

Figure 7-5: Phosphorus G²MPLS testbed: SCN topology.

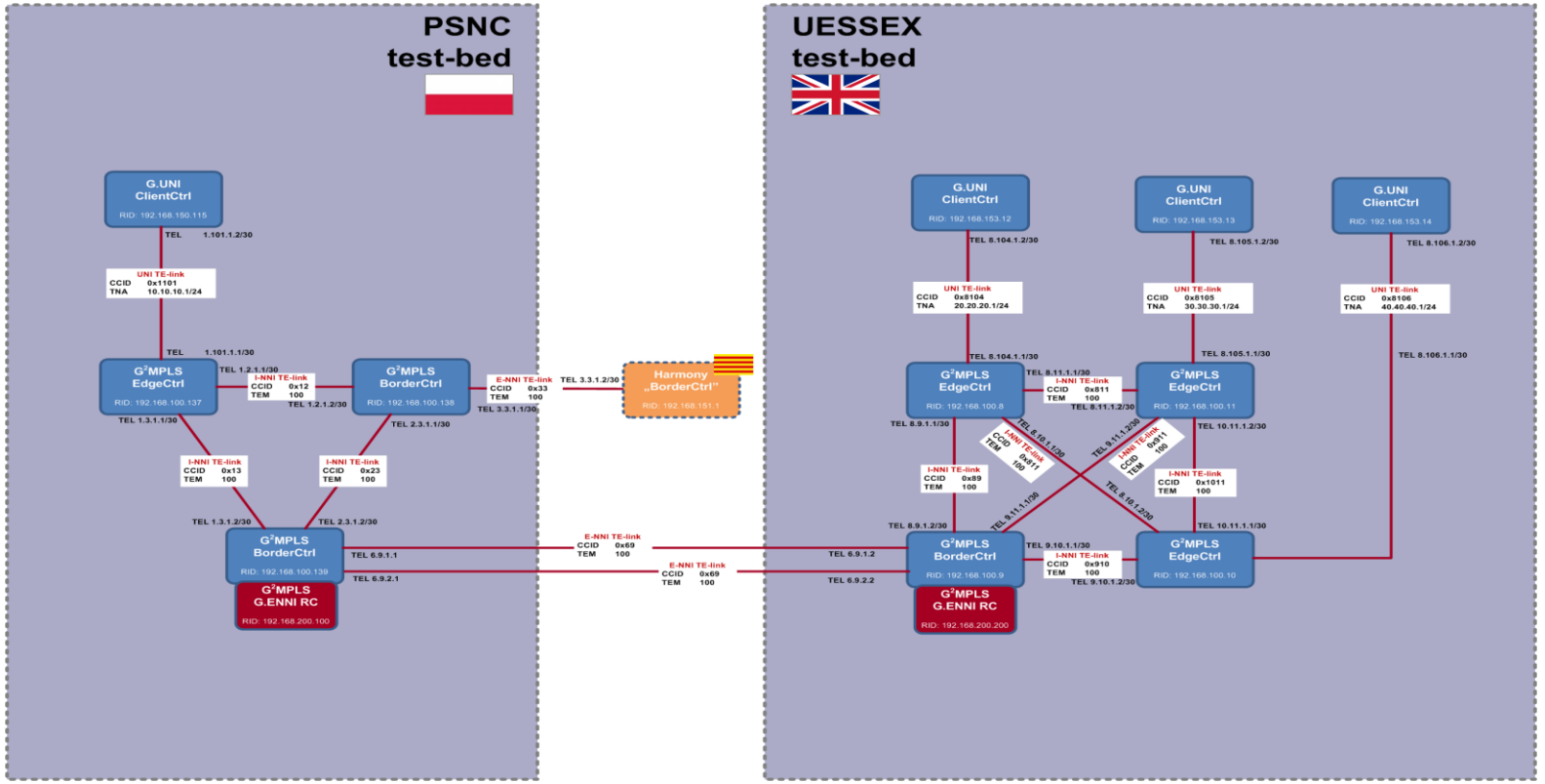

Figure 7-6: Phosphorus G^{2}MPLS testbed: Control Plane logical topology

7.2 G²MPLS public demonstrations overview

The G²MPLS control plane functionalities have been tested in real multi-domain heterogeneous test-bed. This test-bed has also been used for public demonstrations during Supercomputing 2008 (Austin-TX, USA on 16-21 November 2008), ICT'08 (Lyon-FR, EU, on 25-27 November 2008) and TNC2009 (Malaga-SP, EU, on 8-11 June 2009).

The demonstrations held in Supercomputing 2008 and ICT 2008 have been focused on the DDSS application for remote anycast storage of data contents. The demonstration held in TNC2009 has been focused on the KoDaVis application for remote anycast processing and client collaboration. These applications have been interfaced to the G²MPLS Control Plane, and the Phosphorus Integrated model with anycasting has been successfully demonstrated.

An executive description of the G²MPLS capabilities demonstrated in the Phosphorus testbed is provided in the form of a video clip on the Phosphorus website (http://www.ist-phosphorus.eu/documents.php).

7.2.1 SUPERCOMPUTING 2008 event

The Phosphorus WP2 team participated in the SC'08 (Austin-TX, USA) exhibition by successfully demonstrating the capabilities and potentials of the Grid-GMPLS (G²MPLS) Network Control Plane.

In SC'08 WP2 researchers proved the innovative seamless and one-step setup of Grid Network Services (GNS) by integrating a distributed storage grid application (Phosphorus DDSS) with the G²MPLS Control Plane running on the optical Phosphorus pan-European testbed. A numerous and very interested international audience of leading vendors, renowned researchers and executives had the experience of many dynamic transfers of large media contents in a remotely connected storage grid between UK (Univ. Essex labs) and Poland (PSNC labs). The underlying G²MPLS Control Plane implemented dynamically the anycasting connectivity service (i.e. with the dynamic selection of the optimal storage sink by Control Plane) on a DWDM optical infrastructure and GÉANT2 (http://www.geant2.net).

The SC'08 event (http://sc08.supercomputing.org)

The SC Conference is the premier international conference for high performance computing (HPC), networking, storage and analysis. SC08 marks the 20th anniversary of the first SC Conference, then called Supercomputing, held in Orlando, Florida in 1988. Bringing together scientists, engineers, researchers, educators, programmers, system administrators and managers, SC08 is the forum for demonstrating how these developments are driving new ideas, new discoveries and new industries.

7.2.2 ICT 2008 event

The Phosphorus WP2 team participated in the ICT'08 event (Lyon, FR) by continuing the successful demonstration of the numerous capabilities and potentials of the Grid-GMPLS (G^2MPLS) Network Control Plane.

The same demonstration that was held in Supercomputing was repeated in ICT 2008 with the same interest and successful results.

The ICT'08 event (http://ec.europa.eu/information_society/events/ict/2008/index_en.htm)

The biennial ICT Event (formerly called the "IST Event") is the most important forum for discussing research and public policy in information and communication technologies at European level. The Event brings together researchers and innovators, policy and business decision-makers working in the field of digital technologies. The ICT Event is organised by the European Commission's Directorate General for the Information Society and Media and is usually hosted by the current Presidency of the European Union.

7.2.3 Terena Networking Conference 2009 event

The Phosphorus WP2 team participated in the TNC2009 event (Malaga, ES) with successful demonstrations of the Grid-GMPLS (G^2MPLS) Network Control Plane.

Many experts, EU policy makers and NREN directors experienced the innovative G^2MPLS capabilities while controlling ROADMs, fiber and Ethernet switches, all seamlessly integrated with different Grid middleware/applications.

WP2 researchers and other Phosphorus partners proved the seamless and one-step setup of Grid and network resources (Grid Network Services – GNS) through G^2MPLS in two challenging scenarios:

- the remote and distributed storage of large media contents (Phosphorus DDSS) through the optical Phosphorus pan-European testbed;
- the simulation and visualization of complex physical and chemical processes in the atmosphere (KoDaVis application on top of the UNICORE middleware) through an onsite full Gigabit Ethernet testbed.

G^2MPLS Unicast and anycasting connectivity services (i.e. with the dynamic selection of the optimal storage sink or computational element automatically) were demonstrated in both these scenarios and gathered the large interest in the audience.

The TNC2009 event (http://tnc2009.terena.org/)

The TERENA Networking Conference is the major annual European event in research networking, bringing together around 500 participants from many different countries. The 2009 conference has taken place from 7 to

11 June in Málaga, Spain, organised by TERENA and hosted by the University of Málaga and RedIRIS, the Spanish national academic and research network.

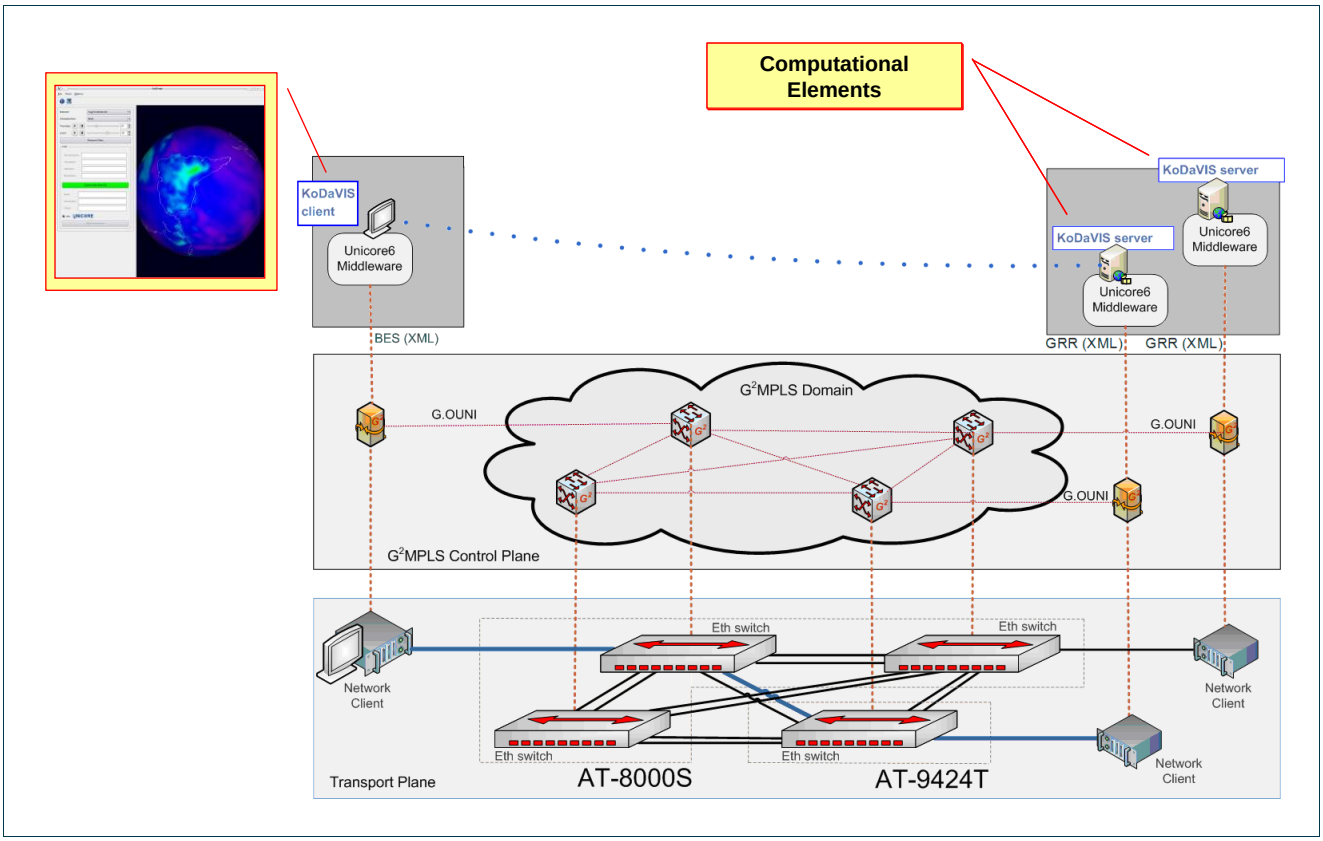

Figure 7-7: Phosphorus G²MPLS testbed for TNC2009.

8 References

The references listed here are only those directly functional to this document. For a list of the references to standards appearing in this document, please point to D2.1, D2.2, D2.3, D2.4, D2.6 and D2.7.

[PH-WP2-D2.1]	Phosphorus deliverable D2.1, “The Grid-GMPLS Control Plane architecture”.
[PH-WP2-D2.2]	Phosphorus deliverable D2.2, “Routing and Signalling Extensions for the Grid-GMPLS Control Plane”.
[PH-WP2-D2.3]	Phosphorus deliverable D2.3, “Grid-GMPLS high level system design”.
[PH-WP2-D2.4]	Phosphorus deliverable D2.4, “Report on Grid-GMPLS Control Plane functional tests”.
[PH-WP2-D2.5]	Phosphorus deliverable D2.5, “Preliminary Grid-GMPLS Control Plane prototype”.
[PH-WP2-D2.6]	Phosphorus deliverable D2.6, “Deployment models and solutions of the Grid-GMPLS Control Plane”.
[PH-WP2-D2.7]	Phosphorus deliverable D2.7, “Grid-GMPLS network interfaces”.
[PH-WP2-D2.8]	Phosphorus deliverable D2.8, “Design of the Grid-GMPLS Control Plane to support the Phosphorus Grid AAI”.
[PH-WP2-D2.9]	Phosphorus deliverable D2.9, “Design of Grid-GMPLS interworking with NRPS”.
[PH-WP2-D2.10]	Phosphorus deliverable D2.10, “Final Grid-GMPLS Control Plane prototype”.
[QUAGGA-DOC]	The Quagga Software Routing Suite documentation. http://www.quagga.net/docs/docs-info.php
[CORBA]	http://www.corba.org/
[omniORB]	http://omniorb.sourceforge.net/

9 Acronyms

AAA	Authentication, Authorisation, and Accounting
AAI	Authentication and Authorization Infrastructure
API	Application Programming Interface
ASON	Automatically Switched Optical Network
BoD	Bandwidth on Demand
BR	Border Router
CE	Computing Element
CORBA	Common Object Request Broker Architecture
CP	Control Plane
CPU	Central Processing Unit
DWDM	Dense Wavelength Division Multiplexing
E-NNI	Exterior NNI
ERO	Explicit Route Object
EU	European Union
G.CR-LDP	G^2MPLS CR-LDP
G.OSPF-TE	GMPLS OSPF-TE
G.OUNI	Grid OUNI
G.OUNI-C	G.OUNI - Client
G.OUNI-N	G.OUNI - Network
G.RSVP-TE	GMPLS RSVP-TE
G²MPLS	Grid-GMPLS (enhancements to GMPLS for Grid support)
GE	Gigabit Ethernet
GÉANT	Pan-European Gigabit Research Network
GLUE	Grid Laboratory Uniform Environment
GMPLS	Generalized MPLS
GNS	Grid Network Service
HW	Hardware
IDM	GÉANT2 Inter-domain Manager
I-NNI	Interior NNI
IP	Internet Protocol
IPR	Intellectual Property Right

IPSec	IP security
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
LAN	Local Area Network
LSP	Label Switched Path
MPLS	Multi Protocol Label Switching
NNI	Network to Network Interface
NREN	National Research and Education Network
OS	Operating System
OSPF	Open Shortest Path First protocol
OSPF-TE	OSPF with Traffic Engineering extensions
O-UNI	Optical UNI
POSIX	Portable Operating System Interface
RC	Routing Controller
RSVP	Resource reSerVation Protocol
RSVP-TE	RSVP with Traffic Engineering extensions
SOAP	Simple Object Access Protocol
SW	Software
TE	Traffic Engineering
TN	Transport Network
TP	Transport Plane
UNI	User to Network Interface
URI	Uniform Resource Identifier
VLAN	Virtual LAN
VM	Virtual Machine
VPN	Virtual Private Network
WP	Work Package
WS	Web Service
XML	Extensible Markup Language

Appendix A G²MPLS VTY commands

A.1 TNRC commands

Command	Description
tnrc	*Start a TNRC instance*
no tnrc	*Stop a TNRC instance*
show tnrc instance	*Show generic TNRC instance info*
show tnrc eqpt	*Show details of configured equipments*
show tnrc board	*Show details of configured boards in the equipment*
show tnrc port	*Show details of configured ports in a board*
show tnrc resource	*Show details of configured resources in a port*
show tnrc xc	*Show details of cross-connections on the equipment*
get-dl-list	*Show list of Data Links ID configured in the TNRC instance data model*
get-dl-details	*Show details of a Data Link configured in the TNRC instance data model*
get-dl-calendar	*Show calendar info of a Data Link configured in the TNRC instance data model*
get-label-list	*Show list of labels related to a Data Link configured in the TNRC instance data model*
get-label-details	*Show details of a label into a Data Link configured in the TNRC instance data model*
equipment	*Set the equipment type to load proper plug-in*
eqpt	*Create an equipment object and load it in the TNRC instance data model*
board	*Create a board object and load it in the TNRC instance data model*

port	*Create a port object and load it in the TNRC instance data model*
resource	*Create a resource object and load it in the TNRC instance data model*
notification eqpt	*Send a notification (synchronization lost, opstate, admstate) for the equipment simulator to the TNRC instance*
notification board	*Send a notification (opstate, admstate) for the board simulator to the TNRC instance*
notification port	*Send a notification (opstate, admstate) for the port simulator to the TNRC instance*
notification resource	*Send a notification (opstate, admstate) for the resource simulator to the TNRC instance*
notification xc	*Send a notification (opstate, admstate) for a cross-connection on the equipment simulator to the TNRC instance*
make-xc	*Request a make cross-connection action to the TNRC instance*
destroy-xc	*Request a destroy cross-connection action to the TNRC instance*
reserve-xc	*Request a reserve cross-connection action to the TNRC instance*
unreserve-xc	*Request an unreserve cross-connection action to the TNRC instance*
load-xc	*Load a cross-connection in the equipment simulator plug-in*

A.2 LRM commands

Command	Description
lrm	*Start an LRM instance*
no lrm	*Stop an LRM instance*
show lrm	*Show generic LRM instance info*
router-id	*Set router ID for the G^{2}MPLS Controller*
scn-if add	*Add a control interface in the LRM instance data model*
scn-if delete	*Delete a control interface in the LRM instance data model*
scn-if enable	*Enable a control interface in the LRM instance data model*
scn-if disable	*Disable a control interface in the LRM instance data model*
show lrm scn-if	*Show details of configured control interfaces*
cc add	*Add a control channel in the LRM instance data model*
cc delete	*Delete a control channel in the LRM instance data model*

cc enable	*Enable a control channel in the LRM instance data model*
cc disable	*Disable a control channel in the LRM instance data model*
cc up	*Set a control channel up in the LRM instance data model*
cc down	*Set a control channel down in the LRM instance data model*
show lrm cc	*Show details of configured control channels*
data-link add	*Add a Data Link in the LRM instance data model*
data-link delete	*Delete a Data Link in the LRM instance data model*
data-link enable	*Enable a Data Link in the LRM instance data model*
data-link disable	*Disable a Data Link in the LRM instance data model*
data-link up	*Set a Data Link up in the LRM instance data model*
data-link down	*Set a Data Link down in the LRM instance data model*
show lrm data-link	*Show details of configured Data Links*
te-link add	*Add a TE Link in the LRM instance data model*
te-link delete	*Delete a TE Link in the LRM instance data model*
te-link enable	*Enable a TE Link in the LRM instance data model*
te-link disable	*Disable a TE Link in the LRM instance data model*
te-link up	*Set a TE Link up in the LRM instance data model*
te-link down	*Set a TE Link down in the LRM instance data model*
te-link bind	*Bind a TE Link to a control channel*
te-link unbind	*Unbind a TE Link from a control channel*
te-link push	*Push a Data Link in a TE Link*
te-link pop	*Pop a Data Link from a TE Link*
te-link set tem	*Set TE Link Metric for a TE Link*
te-link set color	*Set color mask for a TE Link*
te-link set protection	*Set protection type for a TE Link*
te-link set srlgid	*Assign a TE Link to a SRLG*
te-link unset srlgid	*Remove a TE Link from a SRLG*
te-link set tna	*Set TNA id for a TE Link*
te-link set remrcid	*Set remote RC id for a TE Link*

te-link run	*Finalize TE link details in the LRM instance data model and update OSPF*
show lrm te-link	*Show details of configured TE Links*
show lrm adj	*Show details of configured adjacencies*

A.3 SCNGW commands

Command	Description
scngws	*Start a SCNGW server instance*
no scngws	*Stop a SCNGW server instance*
timeout lrm-sync	*Set timeout for synchronization with LRM*
show scngws lrm-conn-status	*Show status of the connection with LRM*
show scngws cleints	*Show details of clients registered on SCNGW server*
show scngws scn-if	*Show details of control interfaces retrieved from LRM*
show scngws cc	*Show details of control channels retrieved from LRM*
show scngws te-link	*Show details of TE Links retrieved from LRM*

A.4 G2.OSPFTE commands

Command	Description
te	*Enable Traffic Engineering extension.*
te on	*Enable Traffic Engineering extension.*
no te	*Disable Traffic Engineering extensions.*
te mpls	*Enable Traffic Engineering for mpls.*
te gmpls	*Enable Traffic Engineering for gmpls.*
te g2mpls	*Enable Traffic Engineering for g2mpls.*
show te router	*Show Traffic Engineering ospf inni instance router information*
show te router-inni	*Show Traffic Engineering ospf inni instance router information*
show te router-enni	*Show Traffic Engineering ospf-enni instance router information*
show te router-uni	*Show Traffic Engineering ospf-uni instance router information*
show te interface	*Shows all interfaces TE information.*

show te interface-inni	*Shows all TE INNI interfaces information. Optional interface can be specified.*
show te interface-enni	*Shows all TE INNI interfaces information. Optional interface can be specified.*
show te interface-uni	*Shows all TE INNI interfaces information. Optional interface can be specified.*
te router-address	*Set IP address of the advertising TE router.*
te router-aa-id	*Configure Associated Area ID.*
te lcl-te-router-id	*Configure Local TE Router ID. This command is obsolete, because LRMD daemon configures router ID for each ospf instance.*
te aa-id	*Configure Associated Area ID.*
te node-ipv4-lcl-prefix add	*Add Node IPv4 Local Prefix.*
te node-ipv4-lcl-prefix clear	*Clear Node IPv4 Local Prefix list.*
te node-ipv6-lcl-prefix add	*Add Node IPv6 Local Prefix.*
te node-ipv6-lcl-prefix clear	*Clear Node IPv6 Local Prefix List*
te reoriginate	*Instant TE-Links re origination. TE-Link owned by ospf instance are reoriginated.*
te force-originate	*Opaques are added to the local database, despite of lack of neighbors.*
te tna force-originate	*TNA opaques are generated and moved to ospf inni instance despite of lack TNA client.*
debug ospf te all	*Enable writing all debug Traffic Engineering information.*
no debug ospf te all	*Disable writing all debug Traffic Engineering information.*
debug ospf te generate	*Enable writing debug Traffic Engineering information about generating opaques.*
no debug ospf te generate	*Disable writing debug Traffic Engineering information about generating opaques.*
debug ospf te originate	*Enable writing debug Traffic Engineering information about originating opaques.*
no debug ospf te originate	*Disable writing debug Traffic Engineering information about originating opaques.*
debug ospf te refresh	*Enable writing debug Traffic Engineering information about refreshing opaques.*
no debug ospf te refresh	*Disable writing debug Traffic Engineering information about refreshing opaques.*
debug ospf te flush	*Enable writing debug Traffic Engineering information about flushing opaques.*
no debug ospf te flush	*Disable writing debug Traffic Engineering information about flushing opaques.*
debug ospf te feed-up	*Enable writing debug Traffic Engineering information about feeding-up opaques.*

no debug ospf te feed-up	*Disable writing debug Traffic Engineering information about feeding-up opaques.*
debug ospf te feed-down	*Enable writing debug Traffic Engineering information about feeding-down opaques.*
no debug ospf te feed-down	*Disable writing debug Traffic Engineering information about feeding-down opaques.*
debug ospf te uni-inni	*Enable writing debug Traffic Engineering information about moving opaques from ospf uni to ospf inni instance.*
no debug ospf te uni-inni	*Disable writing debug Traffic Engineering information about moving opaques from ospf uni to ospf inni instance.*
debug ospf te inni-uni	*Disable writing debug Traffic Engineering information about moving opaques from ospf uni to ospf inni instance.*
no debug ospf te inni-uni	*Disable writing debug Traffic Engineering information about moving opaques from ospf inni to ospf uni instance.*
debug ospf te new	*Enable writing debug Traffic Engineering information about new opaques.*
no debug ospf te new	*Disable writing debug Traffic Engineering information about new opaques.*
debug ospf te delete	*Enable writing debug Traffic Engineering information about deleted opaques.*
no debug ospf te delete	*Disable writing debug Traffic Engineering information about deleted opaques.*
debug ospf te ism-change	*Enable writing debug Traffic Engineering activities when interface state change.*
no debug ospf te ism-change	*Disable writing debug Traffic Engineering activities when interface state change.*
debug ospf te nsm-change	*Enable writing debug Traffic Engineering activities when network state change.*
no debug ospf te nsm-change	*Disable writing debug Traffic Engineering activities when network state change.*
debug ospf te initialization	*Enable writing debug when Traffic Engineering extension is initialized.*
no debug ospf te initialization	*Disable writing debug when Traffic Engineering extension is initialized.*
debug ospf te read	*Enable writing debug when ospf is reading Traffic Engineering information from lrmd daemon.*
no debug ospf te read	*Disable writing debug when ospf is reading Traffic Engineering information from lrmd daemon.*
debug ospf te corba-update	*Enable writing debug when ospf is sending Traffic Engineering information using CORBA interface.*
no debug ospf te corba-update	*Disable writing debug when ospf is sending Traffic Engineering information using CORBA interface.*
debug ospf te corba-set	*nable writing debug when ospf is reading Traffic Engineering*

	information using
no debug ospf te corba-set	*Disable writing debug when ospf is reading Traffic Engineering information using CORBA interface.*

A.5 G2.PCERA commands

Command	Description
g2pcera	*Start a G²PCERA instance*
no g2pcera	*Stop a G²PCERA instance*
g2pcera load topology-file	*Load G²PCERA instance and topologies by file*
show g2pcera	*Show details of a G²PCERA instance*
node add	*Add a node to a G²PCERA instance*
node del	*Delete a node from a G²PCERA instance*
node up network	*Update network node info into G²PCERA instance*
node up grid-site	*Update grid-site node info into G²PCERA instance*
subnode up service	*Add a service subnode to a G²PCERA instance*
subnode up computing-elem	*Add a computing element subnode to a G²PCERA instance*
subnode up sub-cluster	*Add a sub cluster subnode to a G²PCERA instance*
subnode up storage-elem	*Add a storage element subnode to a G²PCERA instance*
subnode del service	*Delete a service subnode from a G²PCERA instance*
subnode del computing-elem	*Delete a computing element subnode from a G²PCERA instance*
subnode del sub-cluster	*Delete a sub cluster subnode from a G²PCERA instance*
subnode del storage-elem	*Delete a storage element subnode from a G²PCERA instance*
tna add	*Add a TNA to a G²PCERA instance*
tna del	*Delete a TNA from a G²PCERA instance*
te-link add	*Add a TE link to a G²PCERA instance*
te-link del	*Delete a TE link from a G²PCERA instance*
update te-link	*Update TE link params (common params, state params, available bw params, srlg list, calendar, ISC list, TDM data, TDM free timeslots, G.709 data, G.709 free data, WDM data, WDM lambda bitmap) in a G²PCERA instance*
g2nscall create	*Create GNS Call*

g2nscall set-info	*Update info (tributary resources, GNS time info, GNS naming info, recovery, LSP info, LSP constraints) into GNS Call*
g2nscall route	*Route GNS Call*
gnscall flush	*Flush GNS Call*
gnscall commit	*Commit GNS Call*
gnscall connection-get	*Get connections for a GNS Call*

A.6 G2.RSVPTE commands

Command	Description
g2rsvpte	*Start a INNI RSVP instance*
no g2rsvpte	*Stop a INNI RSVP instance*
gunirsvp	*Start a UNI RSVP instance*
no gunirsvp	*Stop a UNI RSVP instance*
gennirsvp	*Start a ENNI RSVP instance*
no gennirsvp	*Stop a ENNI RSVP instance*
show g2rsvpte general	*Show generic INNI RSVP instance info*
show gunirsvp general	*Show generic UNI RSVP instance info*
show gennirsvp general	*Show generic ENNI RSVP instance info*
show g2rsvpte interfaces	*Show INNI RSVP instance interfaces retrieved from LRM*
show gunirsvp interfaces	*Show UNI RSVP instance interfaces retrieved from LRM*
show gennirsvp interfaces	*Show ENNI RSVP instance interfaces retrieved from LRM*
show g2rsvpte lsp	*Show INNI RSVP instance LSPs details*
show gunirsvp lsp	*Show UNI RSVP instance LSPs details*
show gennirsvp lsp	*Show ENNI RSVP instance LSPs details*
lsp create	*Create an LSP for INNI RSVP instance*
lsp update	*Update parameters for INNI RSVP instance LSP*
lsp update behaviour	*Update behaviour parameters for INNI RSVP instance LSP*
lsp commit	*Commit INNI RSVP instance LSP*
lsp ero attach	*Attach ERO subobject to INNI RSVP instance LSP*

lsp ero detach	*Detach ERO subobject from INNI RSVP instance LSP*
lps ero commit	*Commit ERO object into INNI RSVP instance LSP*
lsp enable	*Enable INNI RSVP instance LSP*
lsp disable	*Disable INNI RSVP instance LSP*
lsp force-up	*Force signalling up for INNI RSVP instance LSP*
lsp force-down	*Force signalling down for INNI RSVP instance LSP*
lsp send-resv	*Send Resv message for INNI RSVP instance LSP*
lsp send-confirm	*Send Resv Confirm message for INNI RSVP instance LSP*
lsp post-xc-completed	*Post XC completed event for INNI RSVP instance LSP*
lsp destroy	*Destroy INNI RSVP instance LSP*

Appendix B G²MPLS configuration files examples

This appendix presents examples of configuration files for LSC and FSC equipment simulators in LRM and TNRC, which are the core modules in which the users must insert the topology details.

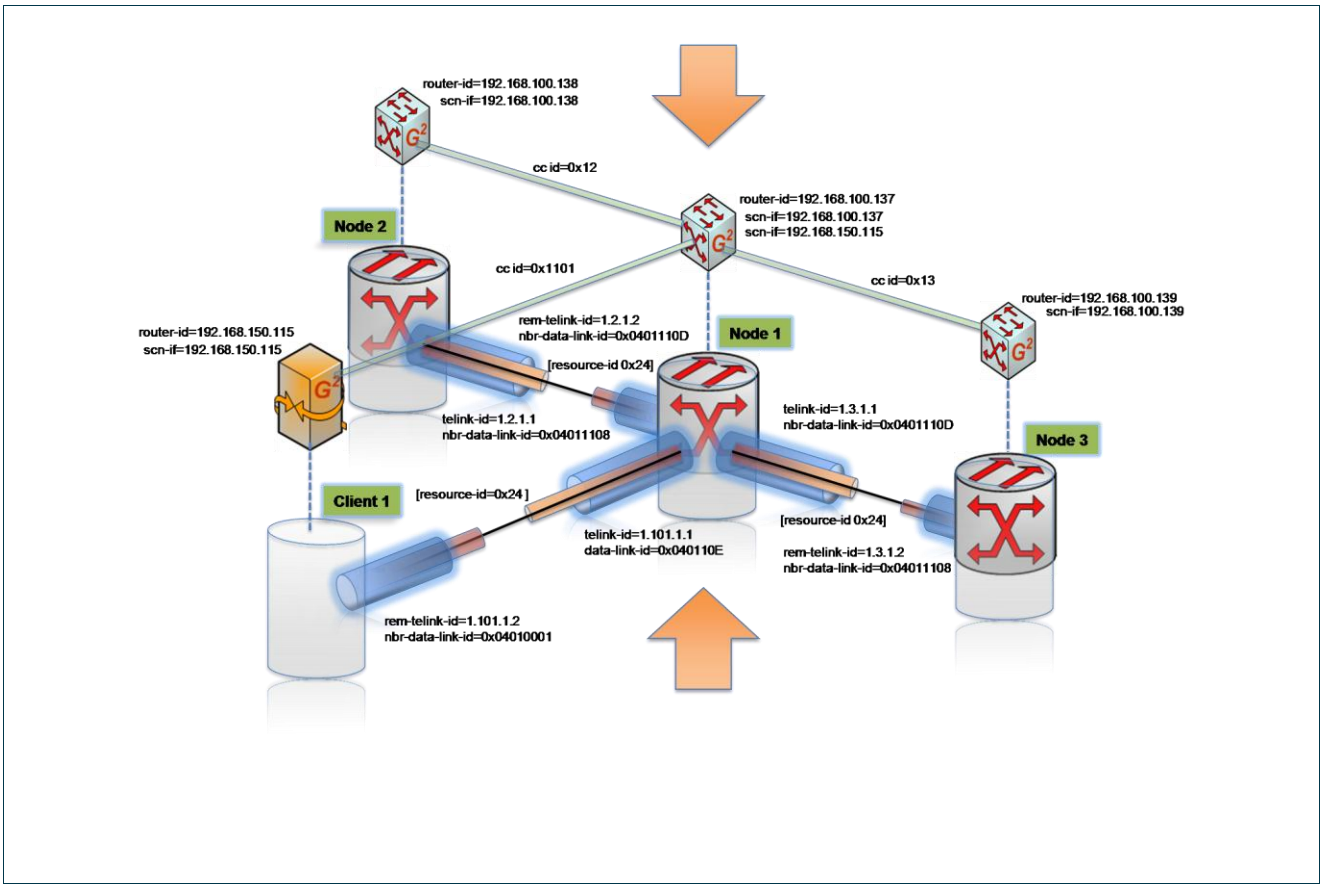

Figure 9-1: Example of G²MPLS controller configuration details (related to Code 9-1 and Code 9-2/Code 9-3)

B.1 LRM configuration

An example of LRM configuration is provided in the following excerpt.

```
!*********NODE 1 - LRM CONFIGURATION FILE*********!
!
!
hostname Node1_lrmd
password zebra
enable password zebra

log file /home/user01/phosphorus-g2mpls/build/var/lrmd.log

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! User specified part (add your commands from here)
!
! starting LRM and entering its context
lrm
!
router-id 192.168.100.137
router-uni-id 192.168.150.137
!
! INNI control interface
scn-if add ip 192.168.100.137
scn-if en ip 192.168.100.137
!
! UNI control interface
scn-if add ip 192.168.150.137
scn-if en ip 192.168.150.137
!
!
! Node1 - Node2
cc add ccid 0x12 scn-ip 192.168.100.137 scn-nbr 192.168.100.138
cc en ccid 0x12
cc up ccid 0x12
!
! Node1 - Node3
cc add ccid 0x13 scn-ip 192.168.100.137 scn-nbr 192.168.100.139
cc en ccid 0x13
cc up ccid 0x13
!
! Node1 TNA
cc add ccid 0x1101 scn-ip 192.168.150.137 scn-nbr 192.168.150.115
cc en ccid 0x1101
cc up ccid 0x1101
!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!! TELink Node1 - Node2 !!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
te-link-inni add ipv4 1.2.1.1 pref-len 30 nbr-id 192.168.100.138 rem-telink-id 1.2.1.2
te-link bind ipv4 1.2.1.1 pref-len 30 ccid 0x12
te-link en ipv4 1.2.1.1 pref-len 30
!
!! DataLinks !!
data-link add id 0x04011108 nbr-id 0x0401110D
data-link en id 0x04011108
```

```
!
!
te-link push ipv4 1.2.1.1 pref-len 30 data-link-id 0x04011108
te-link set ipv4 1.2.1.1 pref-len 30 tem 100
te-link run ipv4 1.2.1.1 pref-len 30
!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!! TELink Node1 - Node3 !!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
te-link-inni add ipv4 1.3.1.1 pref-len 30 nbr-id 192.168.100.139 rem-telink-id 1.3.1.2
te-link bind ipv4 1.3.1.1 pref-len 30 ccid 0x13
te-link en ipv4 1.3.1.1 pref-len 30
!
!! DataLinks !!
data-link add id 0x0401110D nbr-id 0x04011108
data-link en id 0x0401110D
!
te-link push ipv4 1.3.1.1 pref-len 30 data-link-id 0x0401110D
te-link set ipv4 1.3.1.1 pref-len 30 tem 100
te-link run ipv4 1.3.1.1 pref-len 30
!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!! TELink Client TNA  !!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
te-link-uni add ipv4 1.101.1.1 pref-len 30 nbr-id 192.168.150.115 rem-telink-id
1.101.1.2
te-link set ipv4 1.101.1.1 pref-len 30 tna 10.10.10.1 pref-len 24
te-link bind ipv4 1.101.1.1 pref-len 30 ccid 0x1101
te-link en ipv4 1.101.1.1 pref-len 30
!
!! DataLink !!
data-link add id 0x0401110E nbr-id 0x04010001
data-link en id 0x0401110E
!
te-link push ipv4 1.101.1.1 pref-len 30 data-link-id 0x0401110E
te-link run ipv4 1.101.1.1 pref-len 30
```

Code 9-1: A sample lrmd configuration file

B.2 TNRC equipment configuration

Examples of TNRC LSC and FSC equipment configuration files are provided in the following excerpt.

```
!*********NODE 1 - EQUIPMENT SIMULATOR CONFIGURATION FILE*********!!
!
!  Enter in TNRC_NODE
tnrc
!
! EQUIPMENT 1

eqpt id 1 addr 150.254.212.137 type simulator opstate up admstate enabled location XXX

! BOARD 1
board id 1 eqpt-id 1 sw-cap lsc enc-type lambda opstate up admstate enabled
```

```
!

!!!!!!!!!!!!!!!!
! PORT 0x1108
port id 0x1108 board-id 1 eqpt-id 1 flags 0 rem-eq-addr 150.254.212.139 rem-portid
0x110D opstate up admstate enabled bw 0x4CEE6B28 protection unprotected lambdas-base
0x2900000b lambdas-count 40
!
resource id 0x24 port-id 0x1108 board-id 1 eqpt-id 1 type lsc tp-flags 0 opstate up
admstate enabled state free

!!!!!!!!!!!!!!!!
! PORT 0x110D
port id 0x110D board-id 1 eqpt-id 1 flags 0 rem-eq-addr 150.254.212.138 rem-portid
0x1108 opstate up admstate enabled bw 0x4CEE6B28 protection unprotected lambdas-base
0x2900000b lambdas-count 40
!
resource id 0x24 port-id 0x110D board-id 1 eqpt-id 1 type lsc tp-flags 0 opstate up
admstate enabled state free

!!!!!!!!!!!!!!!!
! PORT 0x110E
port id 0x110E board-id 1 eqpt-id 1 flags 0 rem-eq-addr 171.16.50.1 rem-portid 0x0001
opstate up admstate enabled bw 0x4CEE6B28 protection unprotected lambdas-base
0x2900000b lambdas-count 40
!
resource id 0x24 port-id 0x110E board-id 1 eqpt-id 1 type lsc tp-flags 0 opstate up
admstate enabled state free
```

Code 9-2: A sample tnrcd simulator file for Lambda Switching Capable equipment

When adapting above LSC TNRC equipment configuration to FSC configuration, there must be changed switching capability and encoding, and removed all resource descriptions (see Code 9-3).

```
!*********NODE 1 - EQUIPMENT CONFIGURATION FILE*********!
!
!  Enter in TNRC_NODE
tnrc
!
! EQUIPMENT 1

eqpt id 1 addr 150.254.212.137 type simulator opstate up admstate enabled location XXX

! BOARD 1
board id 1 eqpt-id 1 sw-cap fsc enc-type fiber opstate up admstate enabled

!!!!!!!!!!!!!!!!
! PORT 0x1108
port id 0x1108 board-id 1 eqpt-id 1 flags 0 rem-eq-addr 150.254.212.139 rem-portid
0x110D opstate up admstate enabled bw 0x4CEE6B28 protection unprotected lambdas-base 0
lambdas-count 0

!!!!!!!!!!!!!!!!
! PORT 0x110D
```

```
port id 0x110D board-id 1 eqpt-id 1 flags 0 rem-eq-addr 150.254.212.138 rem-portid
0x1108 opstate up admstate enabled bw 0x4CEE6B28 protection unprotected lambdas-base 0
lambdas-count 0

!!!!!!!!!!!!!!!
! PORT 0x110E
port id 0x110E board-id 1 eqpt-id 1 flags 0 rem-eq-addr 171.16.50.1 rem-portid 0x0001
opstate up admstate enabled bw 0x4CEE6B28 protection unprotected lambdas-base 0
lambdas-count 0
```

Code 9-3: A sample tnrcd simulator file for Fiber Switching Capable equipment

<END-OF-DOCUMENT>