034115

PHOSPHORUS

Lambda User Controlled Infrastructure for European Research

Integrated Project

Strategic objective:
Research Networking Testbeds

Deliverable reference number: D.2.5

Preliminary Grid-GMPLS Control Plane

Due date of deliverable: 2007-07-31
Actual submission date: 2007-07-31
Document code: Phosphorus-WP2-D2.5

Start date of project:
October 1, 2006

Duration:
30 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 release notes of the preliminary G^{2}MPLS Network Control Plane prototype. It consists of a set of software modules implementing most of the G^{2}MPLS functionalities designed in Phosphorus WP2, excluding the integrations with AuthN/AuthZ framework as delivered by WP4 and with Network Service Plane (NSP/Harmony) as delivered by WP1.

This deliverable (release notes and software package) constitutes the first official and public release of G^{2}MPLS Control Plane, marked as Milestone M2.4 in the WP2 workplan.

Project:	Phosphorus
Deliverable Number:	D.2.5
Date of Issue:	31/07/08
EC Contract No.:	034115
Document Code:	Phosphorus-WP2-D2.5

List of Contributors/Software Developers

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

Table of Contents

List of Figures

0 Executive Summary

This document provides the release notes of the preliminary G^{2}MPLS Network Control Plane prototype delivered in the form of a XEN virtual machine.

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 configuration process is detailed, in order to set a general reference for any users in his specific deployment scenario.

1 Objectives and Scope

This document briefly describes the contents of the G^{2}MPLS software 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”

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 for the common parts.

This deliverable and its companion deliverable D2.4 on the G^{2}MPLS functional tests represent the first official and public release of the G^{2}MPLS Control Plane.

2 Terminology

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

A full list of the abbreviations used in this document is provided in Section 6.

3 G²MPLS Control Plane preliminary prototype

3.1 Package format

The preliminary G^2MPLS 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 G^2MPLS software modules.

The G^2MPLS 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 G^2MPLS XEN VM consists of two disks

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

The XEN dom0 configuration which is needed to start the G^2MPLS 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: G^2MPLS XEN VM configuration file in dom0.

The G^2MPLS 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.

The

3.2 Package contents

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

- `./build` contains the protocols executables and the common libraries of the G^2MPLS stack; specifically
  - `./build/sbin` groups most of the G^2MPLS protocols executables;
  - `./build/pyg2mpls` groups all the python components (NCC, CCC and framework tools) just described in [PH-WP2-D2.3];
  - `./build/etc` contains some minimal example of configurations and run-scripts;
- `./example` contains an example configuration and start/stop scripts that can be used to run a G^2MPLS core controller with the specified configuration files
  - `./configs` contains the reference configuration files. These files are just a reference and can be modified by the user to implement its own topology for the running G^2MPLS node;
  - `./ctrl_start.sh` and `./ctrl_stop` are two bash scripts used for the start-up and shut-down of the example g2mpls-controller;
  - `./ior` and `./logs` are two directories created during the first execution cycle of the g2mpls-controller and contain, respectively, the CORBA IORs and the saved logs from each G^2MPLS process.

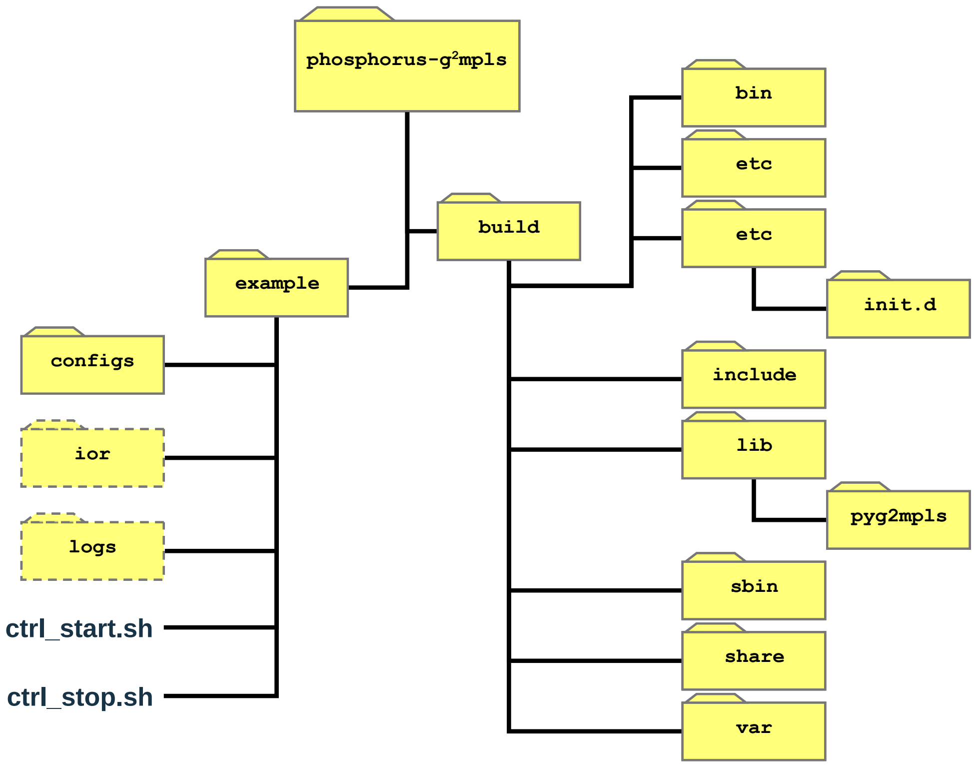

Figure 3-1: Phosphorus G²MPLS prototype build structure.

The G²MPLS 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 G²MPLS 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.

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 G²MPLS controllers (core, border, edge, uniclient) 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 configuation 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
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, uniclient) described in sections

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

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

NOTE. the execution of any xCC modules and of RCD generates persistency files (*.pdb) 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^2MPLS 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^2MPLS core controller

The modules needed by a G^2MPLS 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;
- 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^2MPLS multi-domain topologies.

A G^2MPLS core controller can be run with the init script `build/etc/init.d/g2cored` in which the correct launch sequence is preserved across different start/stop/restart events.

NOTE. In case of failure of any G^2MPLS daemons a full restart of the controller must be done, due to the lack of process dependency tracking in this preliminary release. The final G^2MPLS prototype will fix this issue.

3.3.4 G^2MPLS border controller

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

- tnrcd;
- lrmd;
- scngwsd;
- the python nccd and rcd
- 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^2MPLS multi-domain topologies.
- nccd, i.e. the process implementing the VTY interface to the python xCCs modules

A G^2MPLS border controller can be run with the init script `build/etc/init.d/g2borderd` in which the correct launch sequence is preserved across different start/stop/restart events.

NOTE. In case of failure of any G^2MPLS daemons a full restart of the controller must be done, due to the lack of process dependency tracking in this preliminary release. The final G^2MPLS prototype will fix this issue.

3.3.5 G^2MPLS edge controller

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

- tnrcd;
- lrmd;
- scngwsd;
- the python nccd and rcd
- 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^2MPLS multi-domain topologies.
- nccd, i.e. the process implementing the VTY interface to the python xCCs modules

A G^2MPLS edge controller can be run with the init script `build/etc/init.d/g2edged` in which the correct launch sequence is preserved across different start/stop/restart events.

NOTE. In case of failure of any G^2MPLS daemons a full restart of the controller must be done, due to the lack of process dependency tracking in this preliminary release. The final G^2MPLS prototype will fix this issue.

3.3.6 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;
- the python ccd
- 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.

A G^{2}MPLS G.UNI CLIENT controller can be run with the init script `build/etc/init.d/guniclientd` 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. The final G^{2}MPLS prototype will fix this issue.

4 G^{2}MPLS prototype configuration

The configuration files distributed with the G^{2}MPLS 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.

Most of the Control Plane configurations are contained in the configuration files of tnrcd and lrmd. lrmd acts a hub element for all the protocols, in particular the ospfd and the g2rsvpted. For this reason, the launch sequence of the daemons is crucial and must follow the order:

1. tnrcd
2. lrmd
3. scngwd
4. Python Object Codes
5. g2rsvpted
6. ospfd
7. g2pcerad
8. nccd

Only this order can guarantee the openness of the needed CORBA interfaces and the meaningfulness of the data model information, as explained in sections 3.3.3, 3.3.4, 3.3.5 and 3.3.6.

4.1 Transport Plane resources

The Transport Plane data model is stored in TNRC. The tnrcd.conf file specifies the location of the equipment. This equipment could be a real Transport Plane equipment or a simulator. In the first case, the tnrcd automatically retrieves TN resources from the equipment, while in the latter case (simulator) ports and labels must be declared by the user. The simulator option can be very useful to run the G^2MPLS Control Plane without connecting to real hardware, a feature that can help the training/learning or the pre-production phases.

Examples of tnrcd configuration files for the simulator are provided in the following:

```
!*********EQUIPMENT CONFIGURATION FILE*********!
!
!
tnrc
!
!!!! UNI ports
calient set port 0x04011101 board 1 flags 0 rem-eq-addr 0.0.0.0 rem-portid 0x0 label 0
max-bandwidth 0x4999281A  protection none
calient set port 0x04011103 board 1 flags 0 rem-eq-addr 0.0.0.0 rem-portid 0x0 label 0
max-bandwidth 0x4999281A  protection none

!!!  node-1 --> node 2
calient set port 0x04011201 board 1 flags 0 rem-eq-addr 155.245.93.11 rem-portid
0x04012101 label 0 max-bandwidth 0x4999281A  protection none

!!!  node-1 --> node 3
calient set port 0x04011301 board 1 flags 0 rem-eq-addr 155.245.93.12 rem-portid
0x04013101 label 0 max-bandwidth 0x4999281A  protection none

!!!  node-1 --> node 4
calient set port 0x04011401 board 1 flags 0 rem-eq-addr 155.245.93.13 rem-portid
0x04014101 label 0 max-bandwidth 0x4999281A  protection none
```

Code 4-1: A sample tnrcd simulator file for Fiber Switching Capable equipments.

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

eqpt id 1 addr 10.0.0.1 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 4360 board-id 1 eqpt-id 1 flags 0 rem-eq-addr 10.0.0.2 rem-portid 4365 opstate
up admstate enabled bw 0x4A99CE3D protection unprotected lambdas-base 15 lambdas-count
40
resource id 15 port-id 4360 board-id 1 eqpt-id 1 type lsc tp-flags 0 opstate up
admstate enabled state free
```

```
resource id 20 port-id 4360 board-id 1 eqpt-id 1 type lsc tp-flags 0 opstate up
admstate enabled state free
resource id 25 port-id 4360 board-id 1 eqpt-id 1 type lsc tp-flags 0 opstate up
admstate enabled state free

! PORT 0x110D
port id 4365 board-id 1 eqpt-id 1 flags 0 rem-eq-addr 10.0.0.2 rem-portid 4360 opstate
up admstate enabled bw 0x4A99CE3D protection unprotected lambdas-base 15 lambdas-count
40
resource id 15 port-id 4365 board-id 1 eqpt-id 1 type lsc tp-flags 0 opstate up
admstate enabled state free
resource id 20 port-id 4365 board-id 1 eqpt-id 1 type lsc tp-flags 0 opstate up
admstate enabled state free
resource id 25 port-id 4365 board-id 1 eqpt-id 1 type lsc tp-flags 0 opstate up
admstate enabled state free

! PORT 0x1109
port id 4361 board-id 1 eqpt-id 1 flags 0 rem-eq-addr 10.0.0.2 rem-portid 1 opstate up
admstate enabled bw 0x4999281A protection unprotected lambdas-base 15 lambdas-count 40
resource id 15 port-id 4361 board-id 1 eqpt-id 1 type lsc tp-flags 0 opstate up
admstate enabled state free

! PORT 0x110E
port id 4366 board-id 1 eqpt-id 1 flags 0 rem-eq-addr 10.0.0.2 rem-portid 2 opstate up
admstate enabled bw 0x4999281A protection unprotected lambdas-base 20 lambdas-count 40
resource id 20 port-id 4366 board-id 1 eqpt-id 1 type lsc tp-flags 0 opstate up
admstate enabled state free

! PORT 0x1110
port id 4368 board-id 1 eqpt-id 1 flags 0 rem-eq-addr 10.0.0.2 rem-portid 3 opstate up
admstate enabled bw 0x4999281A protection unprotected lambdas-base 25 lambdas-count 40
resource id 25 port-id 4368 board-id 1 eqpt-id 1 type lsc tp-flags 0 opstate up
admstate enabled state free
```

Code 4-2: A sample tnrcd simulator file for Lambda Switching Capable equipments.

4.2 Control Plane logical topology

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 g2mps_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 the larger than the configuration files of the upper protocols, which inherit most of the information from it.

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

```
! lrmd NODE_1
hostname lrmd_Node1
password zebra
!
log file /tmp/lrmd.log
!
lrm
!
router-id 192.168.90.146
!
scn-if add ip 192.168.90.146
scn-if en ip 192.168.90.146
!
scn-if add ip 131.114.33.146
scn-if en ip 131.114.33.146
!
!
! Node1 - Node2
cc add ccid 1 scn-ip 192.168.90.146 scn-nbr 192.168.90.160
cc en ccid 1
cc up ccid 1
!
! Node1 - Node4
cc add ccid 2 scn-ip 192.168.90.146 scn-nbr 192.168.90.169
cc en ccid 2
cc up ccid 2
!
! Node1 TNA
cc add ccid 7 scn-ip 131.114.33.146 scn-nbr 131.114.33.153
cc en ccid 7
cc up ccid 7
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!! TELink Node1 - Node2 !!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
te-link-inni add ipv4 1.2.1.1 pref-len 30 nbr-id 192.168.90.160 rem-telink-id 1.2.1.2
te-link bind ipv4 1.2.1.1 pref-len 30 ccid 1
te-link en ipv4 1.2.1.1 pref-len 30
!
!! DataLinks !!
data-link add id 0x04011201 nbr-id 0x04012101
data-link en id 0x04011201
!
!
te-link push ipv4 1.2.1.1 pref-len 30 data-link-id 0x04011201
te-link set ipv4 1.2.1.1 pref-len 30 tem 2000
te-link run ipv4 1.2.1.1 pref-len 30
!
!
!
te-link-inni add ipv4 1.2.2.1 pref-len 30 nbr-id 192.168.90.160 rem-telink-id 1.2.2.2
te-link bind ipv4 1.2.2.1 pref-len 30 ccid 1
```

```
te-link en ipv4 1.2.2.1 pref-len 30
!
!! DataLinks !!
data-link add id 0x04011202 nbr-id 0x04012102
data-link en id 0x04011202
!
!
te-link push ipv4 1.2.2.1 pref-len 30 data-link-id 0x04011202
te-link set ipv4 1.2.2.1 pref-len 30 tem 500
te-link run ipv4 1.2.2.1 pref-len 30
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!! TELink Node1 - Node4 !!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
te-link-inni add ipv4 1.4.1.1 pref-len 30 nbr-id 192.168.90.169 rem-telink-id 1.4.1.2
te-link bind ipv4 1.4.1.1 pref-len 30 ccid 2
te-link en ipv4 1.4.1.1 pref-len 30
!
!! DataLinks !!
data-link add id 0x04011401 nbr-id 0x04014101
data-link en id 0x04011401
!
te-link push ipv4 1.4.1.1 pref-len 30 data-link-id 0x04011401
te-link set ipv4 1.4.1.1 pref-len 30 tem 500
te-link run ipv4 1.4.1.1 pref-len 30
!
!
te-link-inni add ipv4 1.4.2.1 pref-len 30 nbr-id 192.168.90.169 rem-telink-id 1.4.2.2
te-link bind ipv4 1.4.2.1 pref-len 30 ccid 2
te-link en ipv4 1.4.2.1 pref-len 30
!
!! DataLinks !!
data-link add id 0x04011402 nbr-id 0x04014102
data-link en id 0x04011402
!
te-link push ipv4 1.4.2.1 pref-len 30 data-link-id 0x04011402
te-link set ipv4 1.4.2.1 pref-len 30 tem 1000
te-link run ipv4 1.4.2.1 pref-len 30
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!! TELink TNA 1 !!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
te-link-uni add ipv4 192.168.10.1 pref-len 30 nbr-id 192.168.90.153 rem-telink-id
192.168.10.2
te-link set ipv4 192.168.10.1 pref-len 30 tna 192.168.1.1 pref-len 32
te-link bind ipv4 192.168.10.1 pref-len 30 ccid 7
te-link en ipv4 192.168.10.1 pref-len 30
!
!! DataLink !!
data-link add id 0x04011101 nbr-id 0x04011102
data-link en id 0x04011101
!
te-link push ipv4 192.168.10.1 pref-len 30 data-link-id 0x04011101
te-link run ipv4 192.168.10.1 pref-len 30
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!! TELink TNA 2 !!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
te-link-uni add ipv4 192.168.11.1 pref-len 30 nbr-id 192.168.90.153 rem-telink-id
192.168.11.2
```

```
te-link set ipv4 192.168.11.1 pref-len 30 tna 192.168.1.10 pref-len 32
te-link bind ipv4 192.168.11.1 pref-len 30 ccid 7
te-link en ipv4 192.168.11.1 pref-len 30
!
!! DataLink !!
data-link add id 0x04011103 nbr-id 0x04011104
data-link en id 0x04011103
!
te-link push ipv4 192.168.11.1 pref-len 30 data-link-id 0x04011103
te-link run ipv4 192.168.11.1 pref-len 30
```

Code 4-3: A sample lrmd configuration file.

5 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-D2.6]	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-D2.3]	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”.
[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/

6 Acronyms

AAA	Authentication, Authorisation, and Accounting
AAI	Authentication and Authorization Infrastructure
ANSI	American National Standards Institute
API	Application Programming Interface
ARGON	Allocation and Reservations in Grid-enabled Optical Networks
ASON	Automatically Switched Optical Network
BB	Bandwidth Broker
BGRP	Border Gateway Reservation Protocol
BoD	Bandwidth on Demand
BR	Border Router
CE	Computing Element
CIM	Computer Integrated Manufacturing
COPS	Common Open Policy Protocol
CORBA	Common Object Request Broker Architecture
CP	Control Plane
CPE	Customer Premises Equipment
CPU	Central Processing Unit
CR-LDP	Constraint-based Label Distribution Protocol
DCM	Distributed Call and Connection Management
DCN	Data Communication Network
DRAC	Dynamic Resource Allocation Controller
DVB	Digital Video Broadcasting
DWDM	Dense Wavelength Division Multiplexing
EGEE	Enabling Grids for E-sciencE
EC	European Commission
EMS	Execution Management Services
E-NNI	Exterior NNI
ERO	Explicit Route Object
ETSI	European Telecommunications Standards Institute
EU	European Union
FCAPS	Fault, Configuration, Accounting, Performance, Security

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
G^2MPLS	Grid-GMPLS (enhancements to GMPLS for Grid support)
GE	Gigabit Ethernet
GÉANT	Pan-European Gigabit Research Network
GGF	Global Grid Forum
GHPN	Grid High Performance Networking
GIS	Grid Information Service
GLUE	Grid Laboratory Uniform Environment
GMPLS	Generalized MPLS
GNS	Grid Network Service
GRAM	Grid Resource Allocation and Management
GSMP	General Switch Management Protocol
HW	Hardware
IANA	Internet Assigned Numbers Authority
IDM	GÉANT2 Inter-domain Manager
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol
I-NNI	Interior NNI
IP	Internet Protocol
IPR	Intellectual Property Right
IPSec	IP security
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
IS-IS	Intermediate System to Intermediate System
ITU	International Telecommunication Union
JSDL	Job Submission Description Language
LAN	Local Area Network
LDP	Label Distribution Protocol
LRMS	Local Resource Management System
LSA	Link State Advertisement
LSDB	Link State Database
LSP	Label Switched Path
LSR	Label Switch Router
MAC	Media Access Control
MAN	Metropolitan Area Network
MP	Management Plane

MPLS	Multi Protocol Label Switching
MPI	Message Passing Interface
NCP	Network Control Plane
NJS	Network Job Supervisor
NMS	Network Management System
NNI	Network to Network Interface
NO	Network Operator
NREN	National Research and Education Network
NRPS	Network Resource Provisioning Systems
NSAP	Network Service Access Point
NSP	Network Service Plane
NTP	Network Time Protocol
OAM	Operations, Administration and Maintenance
OGF	Open Grid Forum
OGSA	Open Grid Services Architecture
OIF	Optical Internetworking Forum
OS	Operating System
OSPF	Open Shortest Path First protocol
OSPF-TE	OSPF with Traffic Engineering extensions
O-UNI	Optical UNI
P2MP	Point to Multi Point
PON	Passive Optical Network
POSIX	Portable Operating System Interface
QoS	Quality of Service
RC	Routing Controller
RFC	Request for Comments
RSVP	Resource reSerVation Protocol
RSVP-TE	RSVP with Traffic Engineering extensions
RTP	Real-time Transport Protocol
SDO	Standard Developing Organizations
SE	Storage Element
SLA	Service Level Agreement
SLS	Service Level Specification
SME	Small and Medium Enterprise
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SP	Service Provider
SPF	Sender Policy Framework
SW	Software
TE	Traffic Engineering
TGC	Trusted Computing Group
TL-1	Transaction Language 1
TLS	Transport Layer Security

TLV	Type-Length-Value protocol fields
TMF	Tele Management Forum
TN	Transport Network
TO	Telecom Operator
TP	Transport Plane
UCLP	User-Controlled Lightpath Provisioning system
UNI	User to Network Interface
UML	Unified Modeling Language
URI	Uniform Resource Identifier
VLAN	Virtual LAN
VM	Virtual Machine
VPN	Virtual Private Network
WAN	Wide Area Network
WG	Working Group
WP	Work Package
WS	Web Service
XML	Extensible Markup Language

<END-OF-DOCUMENT>