034115

PHOSPHORUS

Lambda User Controlled Infrastructure for European Research

Integrated Project

Strategic objective:
Research Networking Testbeds

Deliverable reference number D1.6

Planning and report on functional tests, and prototype for NRPS, Grid GMPLS control plane and middleware interoperability

Due date of deliverable: 2008-03-31
Actual submission date: 2008-03-31
Document code: <Phosphorus-WP1-D1.6v1>

Start date of project:
October 1, 2006

Duration:
30 Months

Organisation name of lead contractor for this deliverable:
Rheinische Friedrich-Wilhelms-Universität Bonn

Revision 1

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

Planning and report on functional tests, and prototype for NRPS, Grid GMPLS control plane and middleware interoperability

Abstract

This deliverable describes setup, execution, and results of tests in the WP1 test network consisting of five interconnected domains, each of them controlled by a separate Network Resource Provisioning System (NRPS).
Furthermore, possibilities for enabling reservations spanning Network Service Plane (NSP) based networks as well as G^2MPLS based networks and their requirements are analysed.

Table of Contents

Table of Figures

0 Executive Summary

This deliverable describes setup, execution, and results of integration tests in the WP1 test network consisting of five interconnected domains: Four domains that are each controlled by a separate Network Resource Provisioning System (NRPS), and one GMPLS domain.

Furthermore, possibilities for enabling reservations spanning Network Service Plane (NSP) based networks as well as G^{2}MPLS based networks and their requirements are analysed.

1 The WP1 testbed

The following subchapters describe both the individual partner testbeds and the data and control plane used for the integration tests of the Network Service Plane. This includes the topology information with the involved hardware, naming spaces and addressing schemes.

1.1 Individual partners testbeds

1.1.1 i2CAT

The switching infrastructure in the i2CAT local testbed is composed by the following equipment:

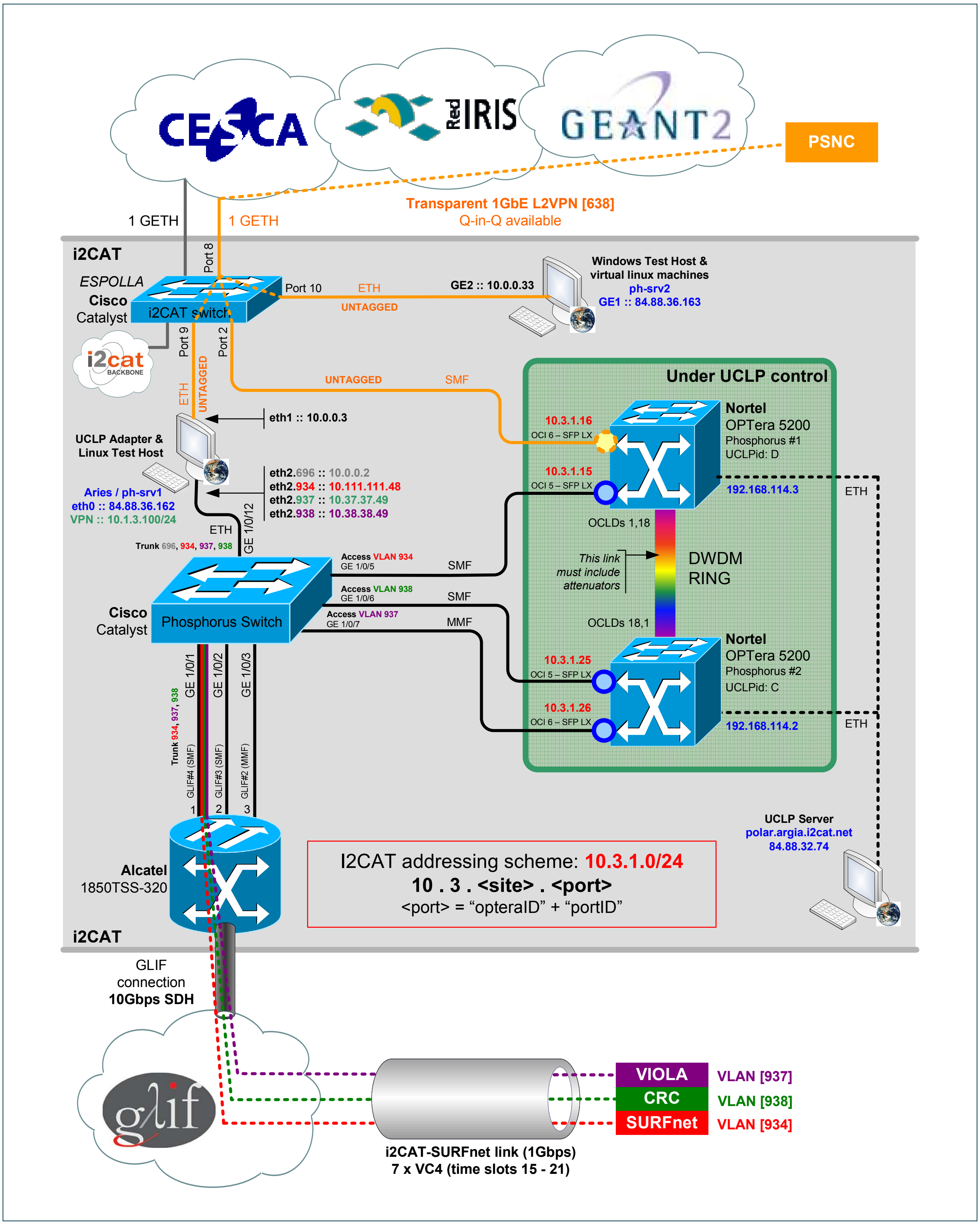
- 1 x Alcatel 1850TSS-320 Transport Service Switch
- 2 x Nortel OPTera Metro 5200 Multiservice Platform
- 2 x Cisco Catalyst Gigabit Ethernet Switch

The Alcatel 1850TSS-320 is a multiservice node, with a commutation capacity up to 320Gbps and with matrix, control units and power supply redundancy. The applications for which it is designed are MSPP (Multi Service Provisioning Platform), MSTP (Multi Service Transport Platform) and it also can be used as Ethernet/RPR/MPLS aggregation node. It supports several protocols for packet and circuit traffic. It also counts with several mechanisms of QoS and failure tolerance. This device includes an agnostic matrix (universal) which is able to commute indistinctively frames (Ethernet, RPR, MPLS) and circuits (SDH/SONET/LCAS/VCAT), as well as to incorporate support for WDM and OTN. Inside the testbed, this node is used to multiplex/demultiplex 3 VLANs (7xVC-4 each one) received through a 10Gbps SDH link connected to the GLIF Infrastructure. These three VLANs are forwarded and sent each one of them to a port of a switch, connected to the OPTeras core and the application switch.

The Optical Metro 5200 is a DWDM multi-service platform that delivers 32 wavelengths of bandwidth, 10G scalability and sub-rate multiplexing capabilities that simplify the deployment of efficient DWDM metropolitan networks. The open and modular architecture of the Optical Metro 5200 delivers network scalability, per-wavelength manageability, bit-rate and protocol independence, and ring survivability. Up to M18, these devices were connected with each other, creating a DWDM ring. These two devices are the ones that are be configured by UCLPv2 via TL1, offering several interconnection possibilities between the connected partners and the local applications. I2CAT is studying the feasibility of introducing Ethernet switching equipment to make the testbed more robust and widen the switching options available.

The Cisco Catalyst switches are used to commute the VLANs received from the remote partners and to offer connection to the local applications. These switches currently have static configurations but it may be possible to configure them with UCLP/ARGIA in the near future. All of them are connected to the OPTera core network that will be the one that will enable/disable the connections.

I2CAT has improved its testbed by introducing test hosts which can test simultaneously both, Géant2 and GLIF links to other WP1 partners. Furthermore, they allow testing the local testbed itself, as the hosts are connected both at the entrance and the way out of the testbed.



Project:	Phosphorus
Deliverable Number:	D1.6
Date of Issue:	31/03/08
EC Contract No.:	034115
Document Code:	<Phosphorus-WP1-D1.6v1>

Figure 1.1: The I2CAT local testbed.

1.1.2 SURFnet

The SURFnet testbed that is used for testing of DRAC consists of an environment that is technologically equivalent to the larger SURFnet network. It can, as part of the Phosphorus project, also be used to connect ports for applications provided by local partners (SARA, UvA) to the actual middleware-controlled network before these ports go on the production network.

The testbed currently contains three Nortel OME6500s and two servers running a DRAC controller – one in the public network, and one for internal test purposes. The SURFnetOME testbed is connected with three ports to the SURFnet L2 Phosphorus hub switch, making it a possibility to connect directly to the three other partners in WP1, and when required to any other partner wishing to make use of the SURFnet testbed.

The connections across the testbed can be controlled manually by the participants via the web interface at http://drac.surfnet.nl/ . Additionally the web services interface that is used to interface with the NSP-Adapter is also available. The NSP Adapter itself runs on a server at SURFnet and is accessible via a VPN.

The OME testbed switches VC4 connections when requested by DRAC. The three nodes are connected via 10 Gbps links, building a triangle. Each of the nodes has several GbE ports and is able to switch each GbE port to any other GbE port. Three GbE ports are connected to the Phosphorus hub switch where transparent and static VLAN-based connections have been created towards the three other WP1 partners.

Next to the GbE edge points there is also a L1 STM-64 connection to NetherLight that can be controlled. This connection is physically connected between the testbed and NetherLight and makes it trivial to expand testing to include multiple heterogeneous technologies.

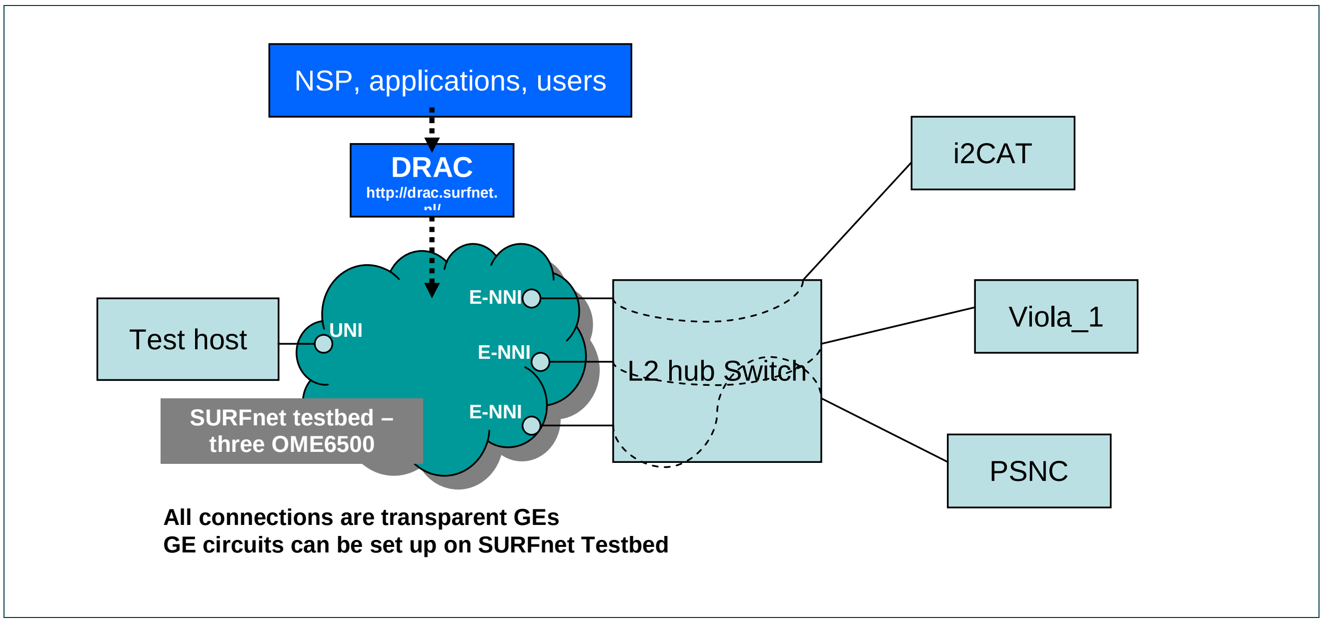

NSP, applications, users
DRAC
http://drac.surfnet.nl/
Test host
UNI
E-NNI
E-NNI
E-NNI
SURFnet testbed – three OME6500
L2 hub Switch
i2CAT
Viola_1
PSNC
All connections are transparent GEs
GE circuits can be set up on SURFnet Testbed

Figure 1.2: The SURFnet local testbed (abstract topology).

1.1.3 VIOLA

1.1.3.1 *Testbed topology*

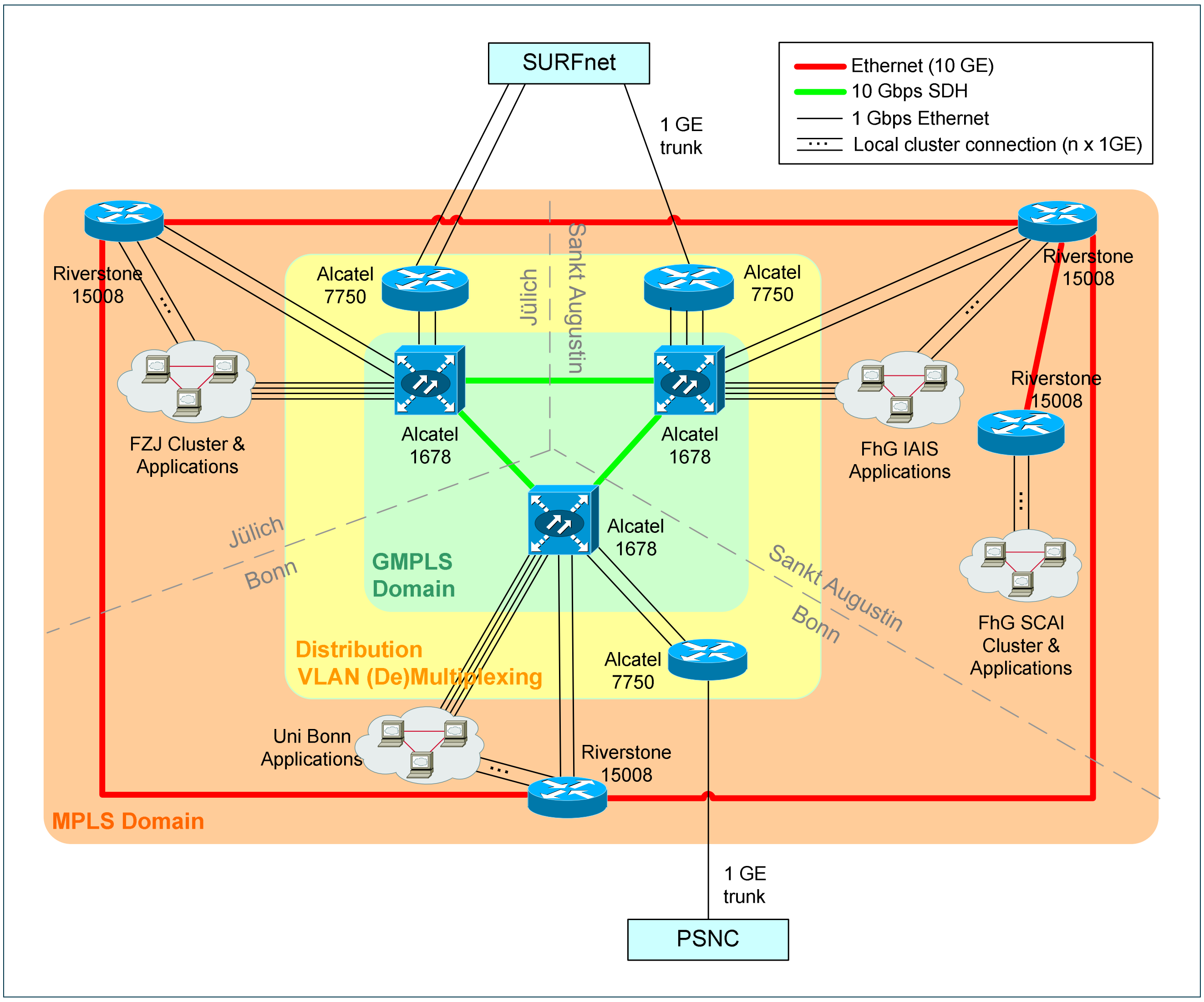


Figure 1.3: The VIOLA local testbed.

The VIOLA testbed consists of three sites, located in Bonn (University of Bonn), Jülich (Forschungzentrum Jülich) and St. Augustin (Fraunhofer IAIS, Fraunhofer SCAI). These sites are interconnected with MPLS as well as GMPLS equipment.

Figure 1.3 depicts the VIOLA testbed. The physical interdomain links arrive at Alcatel 7750 devices whose task is to multiplex different VLANs, each of which appears as an own interdomain link to the NRPSs operated in the VIOLA domain, to a single physical interdomain link. At each location, an Alcatel 1678 device belonging to the

GMPLS domain is connected to the Alcatel 7750. Between these devices, each VLAN is assigned a separate link.

The VIOLA MPLS domain consists of three Riverstone 15008 routers. It is controlled by the ARGON NRPS, connected to the central NSP instance by an ARGON Adapter. Both the ARGON core and the ARGON Adapter are operated at the University of Bonn.

The VIOLA GMPLS domain consists of three Alcatel 1678 devices. It is controlled by the Thin NRPS developed in the Phosphorus project. Until the end of 2007, the Thin NRPS was operated at FhG-IAIS in St. Augustin. From 2008 on, it is operated at the University of Bonn. This domain is part of the testbed to show interoperability of the Network Service Plane with a GMPLS controlled domain.

At each of the VIOLA sites, the MPLS and the GMPLS are connected by two interdomain links, making a total of 6 interdomain links.

1.1.3.2 *Addressing scheme*

The TNAs are configured in such a way that the third octet shows the site at which the TNAs are located. This is achieved by defining three location IDs 1 (Bonn), 2 (Jülich), and 3 (Sankt Augustin). In the GMPLS domain, the third octet of a TNA is set to the corresponding location ID. In the MPLS domain, a preceding "1" is added. As there is a second Riverstone 15008 MPLS device located in Sankt Augustin, a trailing "2" is added to the third octet for this device.

A summarization of all TNA address spaces in the VIOLA domain is shown in Table 1.1.

Site	Site number	Netmask
Bonn	1	10.7.1.0/24 (GMPLS) 10.7.11.0/24 (MPLS)
Jülich	2	10.7.2.0/24 (GMPLS) 10.7.12.0/24 (MPLS)
Sankt Augustin	3	10.7.3.0/24 (GMPLS) 10.7.13.0/24 (MPLS) 10.7.132.0/24 (MPLS, 2nd Router)

Table 1.1: Addressing scheme within the VIOLA testbed

1.1.4 CRC

The CRC testbed comprises equipment located in the Broadband Applications and Demonstration Laboratory (BADLAB^{TM}) and the Optical Networking Laboratory (ONL) facilities and extends across CANARIE, Canada's advanced network. The CANARIE network elements assigned to the Phosphorus project can be managed and controlled via the UCLP NRPS associated with the CRC domain.

A total of 6 network elements (5 Nortel OME 6500s and one Nortel HDXc) across CANARIE can be managed and controlled by the UCLP-CRC NRPS to manipulate the four LightPaths (LPs) allocated to the Phosphorus project. The Nortel Optical Multiservice Edge (6500) and Optical Cross Connect (HDXc) platforms support SONET switching and have OC-192 interfaces and line cards with dedicated GbE ports. All LPs are based on STS-24c and STS3c-7v channels with the end points facing the CRC BADLAB facility terminated on GbE ports which connect to a Cisco 3750 switch co-located with the CANARIE Ottawa POP. The Cisco 3750 switch is connected to the BADLAB's core switch/router, a Cisco Catalyst 6509, via an 8-channel CWDM infrastructure. A Nortel OPTera Metro 5200 is also available as part of the Optical Networking Laboratory (ONL) adjacent to the BADLAB and supports GbE interfaces which can be connected to the Catalyst 6509 and/or PC's.

L2 switches are also used to multiplex and demultiplex VLANs on the connections to other local testbeds to simulate multiple parallel links. These switches also allow for creating connections (VLANs) between local testbeds which are not directly connected – they can be connected via PSNC and SURFnet using a direct VLAN.

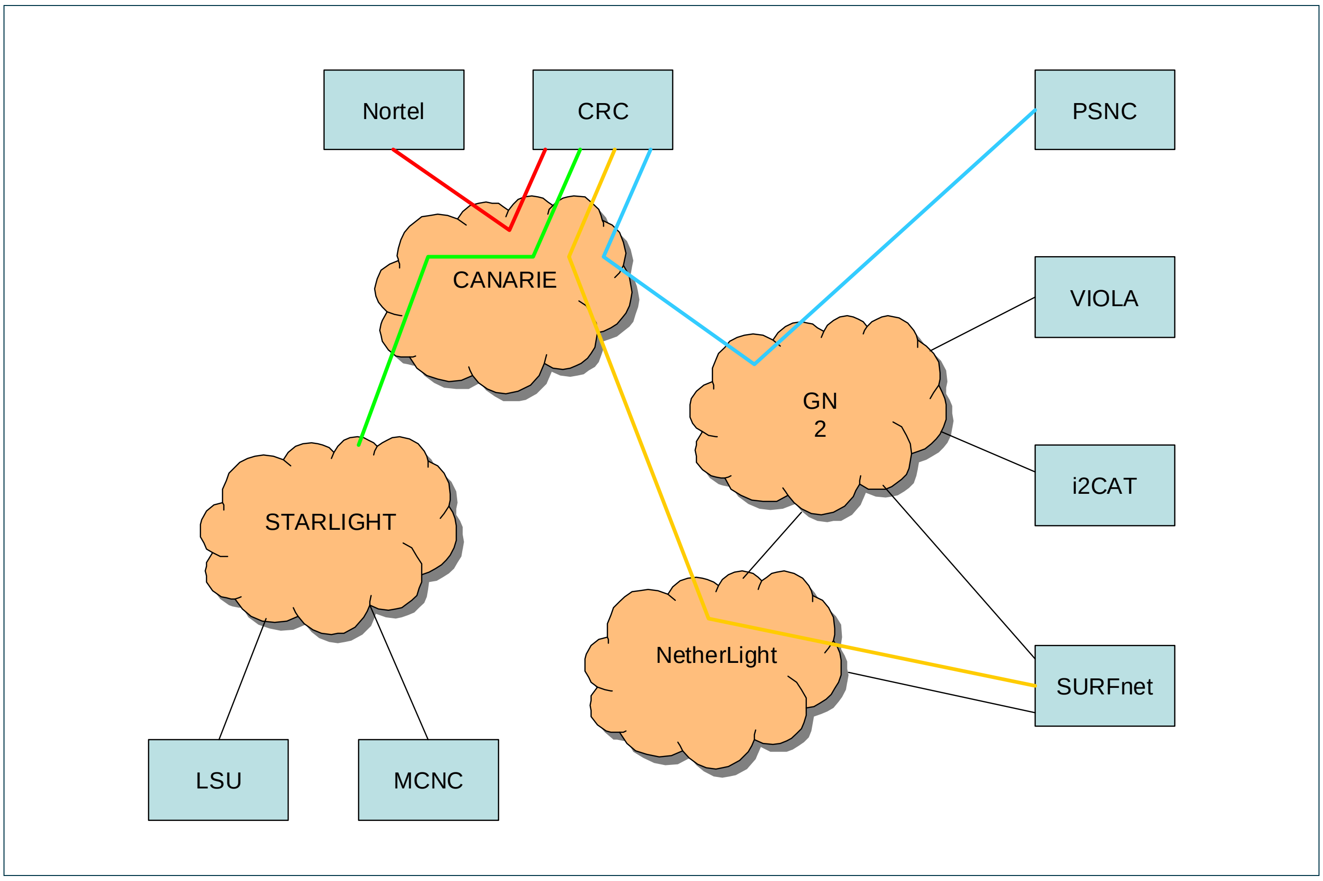


Figure 1.4: CRC Testbed Connectivity to Other Project Partners (cf. Figure 1.5 for details on coloured lightpaths)

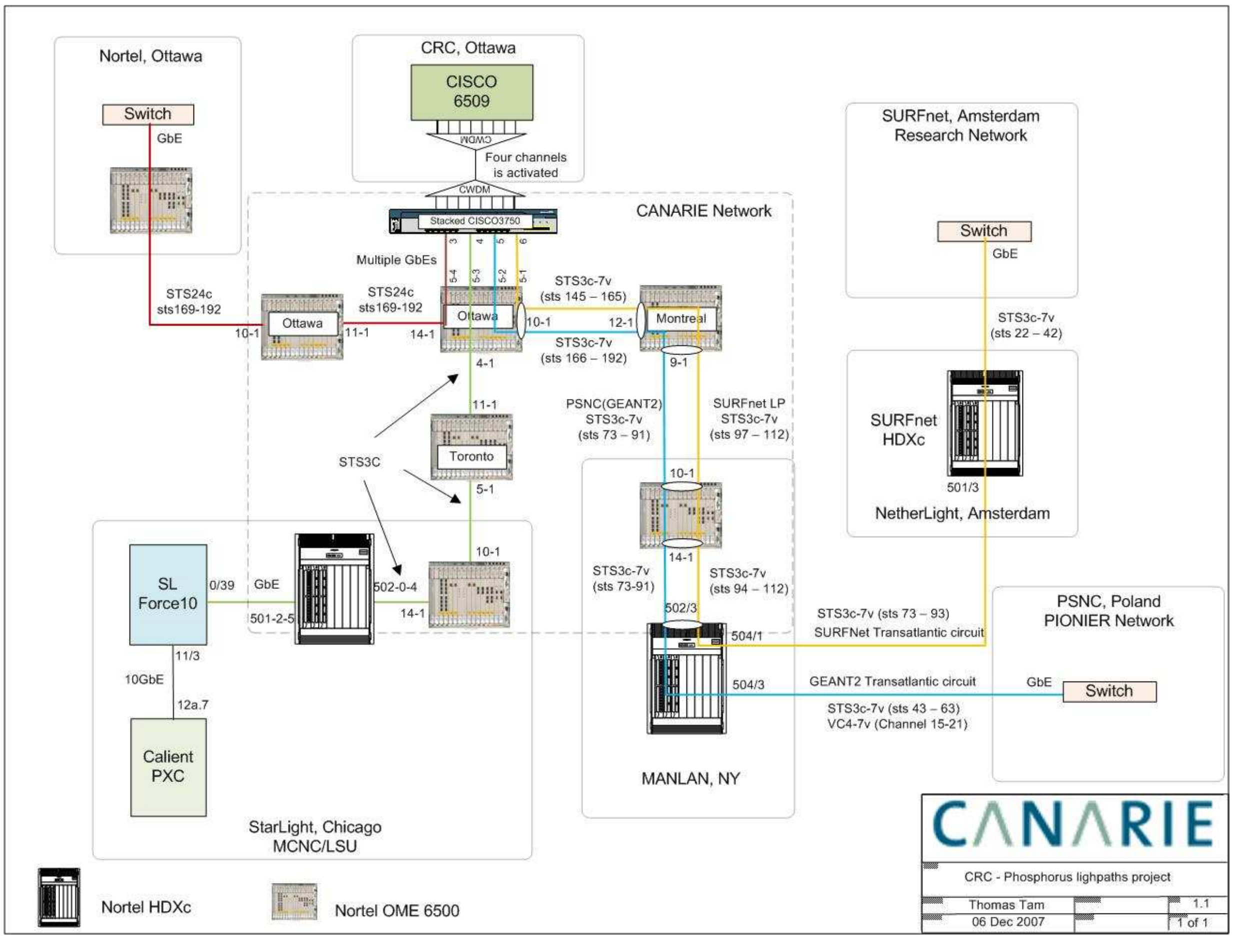


Figure 1.5: Details of CRC Phosphorus LPs

Project:	Phosphorus
Deliverable Number:	D1.6
Date of Issue:	31/03/08
EC Contract No.:	034115
Document Code:	<Phosphorus-WP1-D1.6v1>

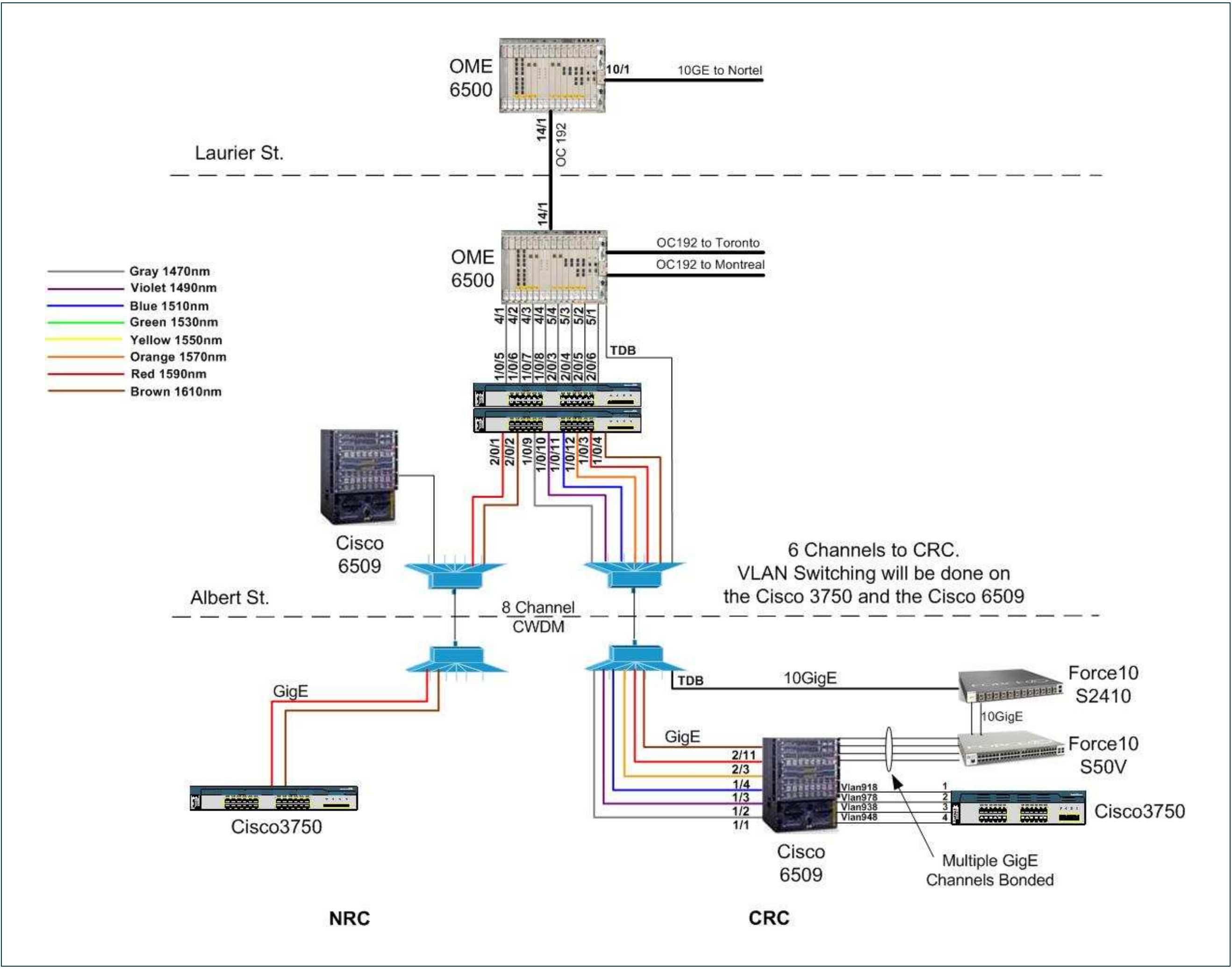


Figure 1.6: CRC’s Local Testbed Connectivity Infrastructure

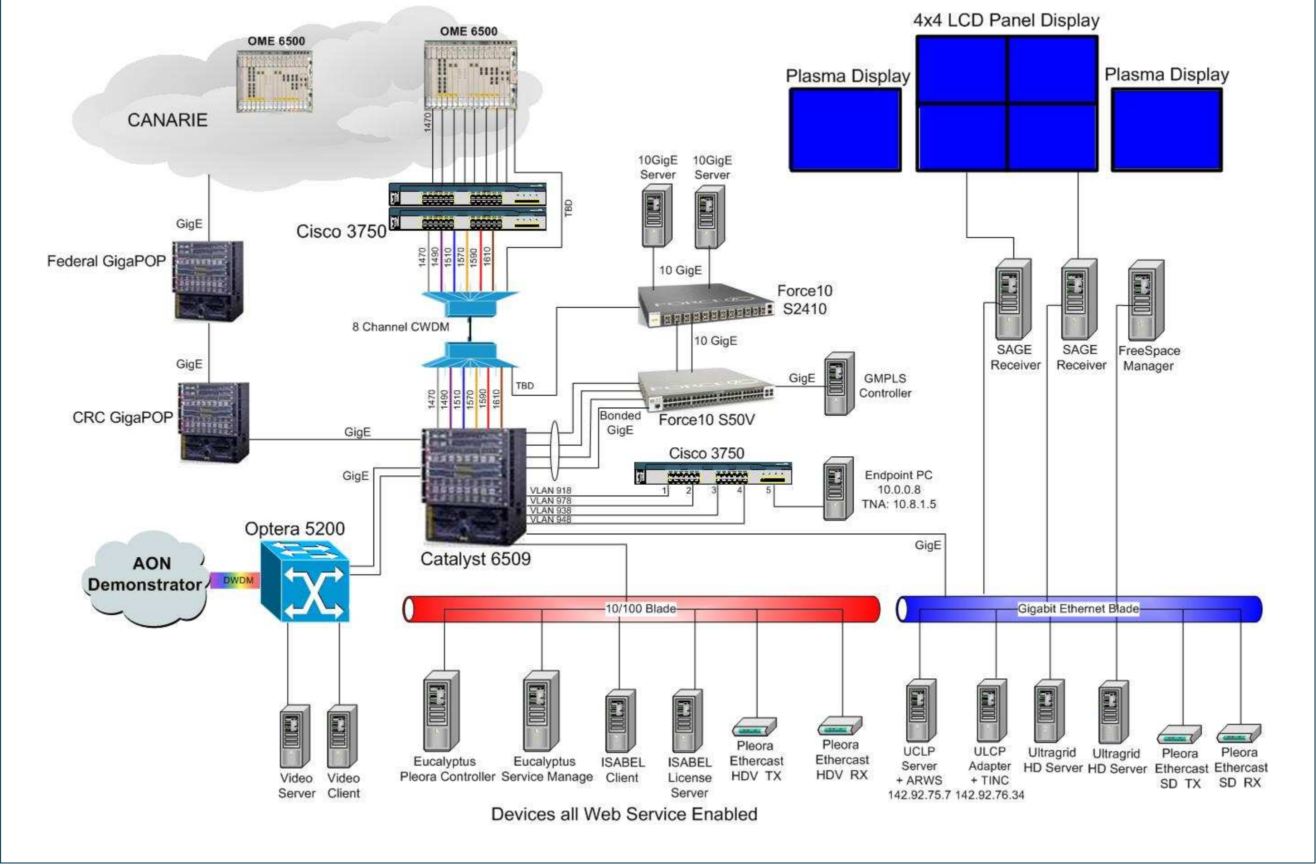


Figure 1.7: Detailed View of CRC's Local Testbed

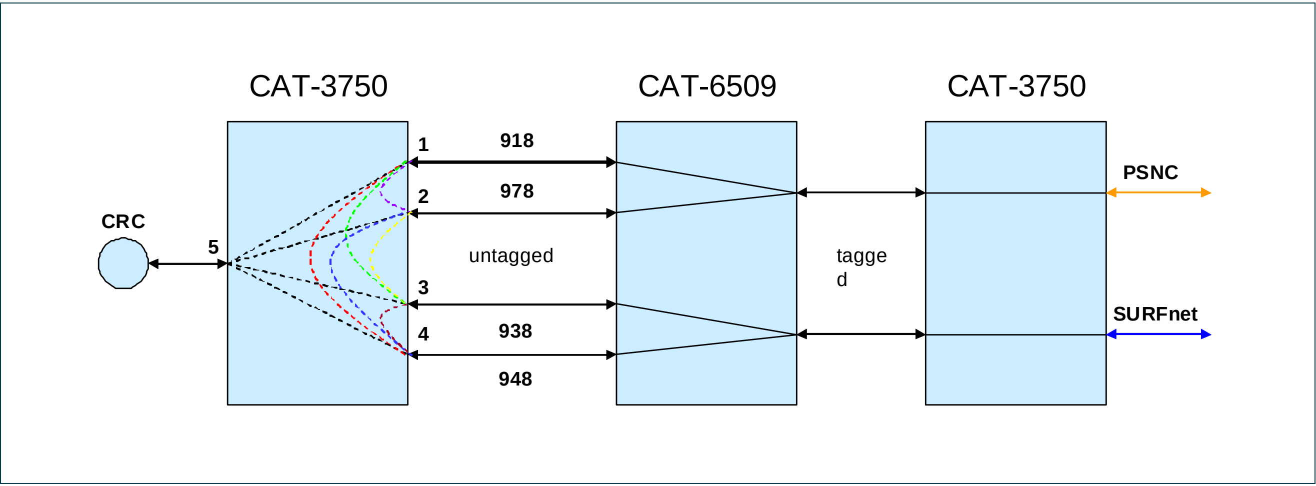


Figure 1.8: CRC Domain Controlled via UCLP (switching of VLANs)

1.2 Interdomain data plane configuration

1.2.1 Interdomain connections: VLAN naming spaces

The different domains within the WP1 data plane are connected either directly, like the different VIOLA domains, or via L2VPNs. Interdomain links based on L2VPNs use either Géant2 point-to-point or connections within GLIF infrastructure.

The domain numbering convention was defined and agreed as follows:

Domain/Site numeric identifier	Domain Name
1	PSNC
2	*not used (CESnet)*
3	I2CAT
4	SURFnet
5	SARA
6	UESSEX
7	VIOLA
8	CRC

Table 1.2: Numbering convention for domain/site identifiers in Phosphorus testbed.

The VLAN numbering scheme follows a pattern based on an ordered combination of the two domain identifiers. That is to say, two linked domains, if connected via tagged L2VPN, will generate a VLAN identifier constructed from a numeric prefix plus a combination of the numeric identifiers of each one of the domains. If several VLANs link two domains, a 1-digit prefix will be added to avoid VLAN identifier duplication.

Let X be the decimal, 1-digit, convened prefix of the testbed; Y, Z the numeric identifiers for two different domains and n the 1-digit prefix for duplication avoidance (n valued between zero and nine, both included). Then, VLANs between these domains will be tagged as either nXYZ or nXZY. It is important to highlight that ordering of domain identifiers matters. As a consequence, the maximum number of VLAN identifiers between two domains is 20, given a fixed prefix X.

Currently, the VLANs configured in WP1 testbed are:

VLANs IDs	CRC	I2CAT	SURFnet	VIOLA
VIOLA	978	937	947	
SURFnet	948	934		
I2CAT	938			
CRC				

Table 1.3: VLAN identifiers used in the testbed provided by WP1 members

Furthermore, other VLANs have been configured for connecting remote clients located in other partners' premises, such as PSNC, University of Essex or SARA:

• PSNC (ID=1) to VIOLA-GMPLS (ID=7):	VLANs 717, 817, 917, 2817, 2917
• UESSEX (ID=6) to VIOLA-GMPLS (ID=7):	VLAN 1967
• SARA (ID=5) to VIOLA-GMPLS (ID=7):	VLAN 957

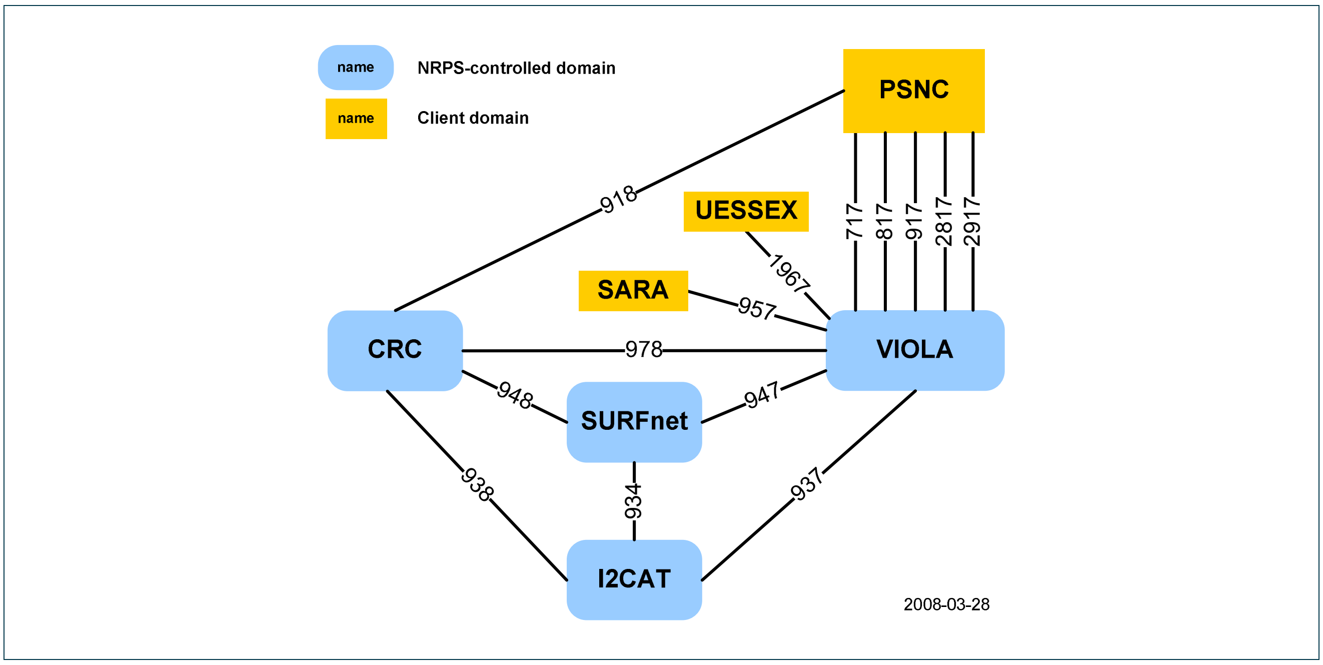


Figure 1.9: VLAN map and addressing.

1.2.2 Data plane addressing scheme

WP1, in conjunction with WP6, has followed own addressing schemes for the testbed as described in this section. As WP1 software prototypes deal with layer 2 resource provisioning systems, an addressing scheme has been proposed and convened among all partners to identify endpoints uniquely within the testbed. This addressing scheme is based on the numeric domain identifiers exposed before and follows an IPv4-like pattern, that is, every domain has its own TNA space with its own mask.

The endpoints belonging to the different domains are identified as follows:

Domain/Site numeric identifier	Domain Name	TNA space (address / mask)	Subspaces used	
1	PSNC	*Does not apply*	*Does not apply*	
3	I2CAT	10.3.1.0 / 24	UCLP	10.3.1.0 / 24
4	SURFnet	10.4.1.0 / 24	DRAC	10.4.1.0 / 24
5	SARA	*Does not apply*	*Does not apply*	
6	UESSEX	*Does not apply*	*Does not apply*	
7	VIOLA	10.7.0.0 / 16	GMPLS	10.7.0.0 / 21
			ARGON	10.7.8.0 / 21 10.7.128.0 / 21
8	CRC	10.8.1.0 / 24	UCLP	10.8.1.0 / 24

Table 1.4: Phosphorus TNA addressing scheme.

With respect to IP addressing, WP1 partners have defined an own IP addressing for test hosts based on private IP addresses within the range of the network address 10.0.0.0 with mask 255.255.255.0.

The tests hosts used regularly are compiled in the following list:

• CRC:	10.0.0.8
• SURFnet:	10.0.0.4
• I2CAT:	10.0.0.3, 10.0.0.33
• UniBonn:	10.0.0.71
• FHG:	10.0.0.73
• FZJ:	10.0.0.72

In **Figure 1.10**, a full map of the Phosphorus testbed can be found, in which VLAN tagging, TNA naming and IP addressing are shown.

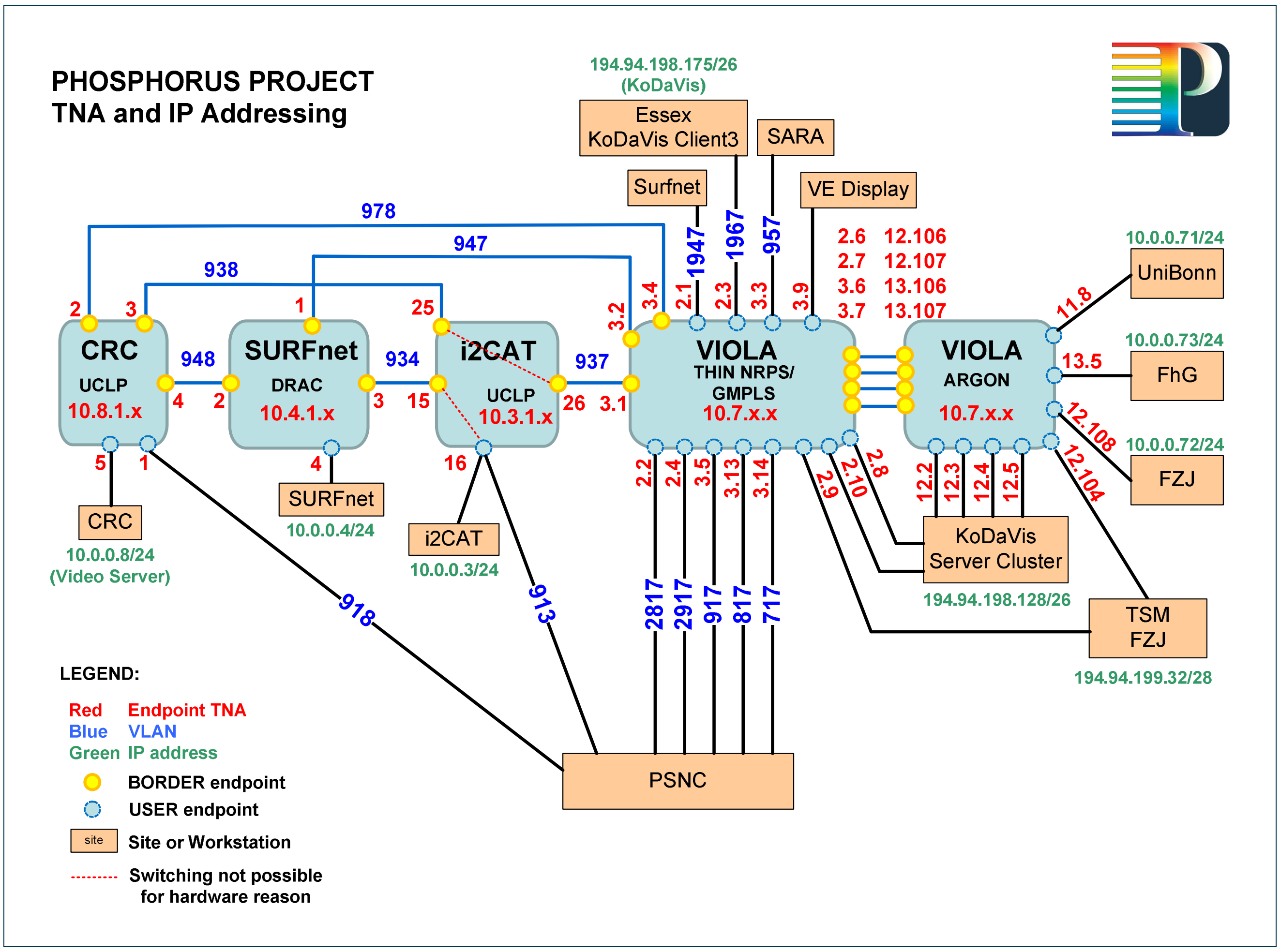


Figure 1.10: Full Phosphorus map of the testbed with VLAN, TNA and IP addressing.

1.3 Interdomain control plane configuration

Signalling between the domains participating in the WP1 testbed consists of web service calls between the NRPS Adapters and the central NSP instance. As described in detail in [Phosporus-D1.4], the NRPS Adapters register by calling the addDomain and editDomain operations of the central NSP instance's Topology Web Service (Topology-WS), and the central NSP reserves resources in the different domains by calling the isAvailable and createReservation operations of the corresponding NRPS Adapters' Reservation Web Service (Reservation-WS).

A web service is identified by an *Endpoint Reference* (EPR) that contains the host name or IP address of the server running the web service. In the WP1 testbed, the control plane is not coupled with the data plane. Instead, the regular Internet connectivity is used for signalling between the different systems. To secure the testbed against unauthorized access from the Internet (cf. Section 4.4), a VPN has been set up based on the tinc software [tinc], following the recommendations of WP6 (cf. [Phosphorus-D6.1]).

The address scheme proposed in by WP6 has been adopted: The first octet of the IPv4 address is set to the decimal value 10, indicating that this is a private IP address (cf. [RFC1918]). The second octet is set to 1, indicating that this is a WP1 testbed address. The third octet is set to the number associated with the project partner. The fourth octet is assigned to different systems by the project partner hosting these systems.

Table 1.5 shows all control plane addresses currently in use in the WP1 testbed. The central NSP instance is maintained and hosted by the University of Bonn, therefore its VPN IP address is located in the VIOLA subnet.

Local testbed	Domain name	VPN IP address
I2cat	`i2CAT`	10.1.3.100
Surfnet	`surfnet-testbed`	10.1.4.1
VIOLA	*(central NSP instance)*	10.1.7.1
	`viola-mpls`	10.1.7.2
	`viola-gmpls`	10.1.7.3
CRC	`CRC`	10.1.8.1

Table 1.5: Control plane addresses within the WP1 testbed

2 Test plan

The subsequent chapters describe the different components and the workflow to run the integration test of the Network Service Plane with each participating system. On the one hand, this includes testbed initialization, connection establishment, path usage, and path teardown and, on the other hand, the involved data structures and tools on the other side. The corresponding test results are then documented in Chapter 3.

2.1 Components

For the integration test phase of the Network Service Plane all individual software modules are combined and tested as a group. To fulfil the test requirements the whole system has to pass different test components. These components are described in this chapter and include blocks for testbed and connection tests.

2.1.1 Testbed initialization

The initialization of the testbeds has been implemented in a way that makes it easy for a domain to be part of the controlled resources. Using an automatic registration process and several simple configuration files, a domain can register itself and all its network resources in the NSP. After the automatic registration, a domain is ready to be fully controlled by the NSP. The next subsections explain in full detail the initialization process of a domain and its testbed.

2.1.1.1 *Domain registration*

The NSP needs some information about a domain and its network resources (the testbed topology) in order to control it. This information is taken from the NRPS Adapter in a passive way, i.e. it is not the NSP who requests the information, but the NRPS Adapter itself that sends automatically the information to the NSP. Thus, the NRPS Adapter mainly translates requests from the NSP to the NRPS and vice-versa. It also performs actions such as retrieve topology from NRPS and send it upwards towards the NSP. When the information is received in the NSP, it stores it in its database. Obviously, a domain can only register with an NSP if this last one is up and running, otherwise, the NRPS Adapter will get a *ConnectionRefused* error and the registration can not be performed.

To implement the registration in a common way, each NRPS Adapter extends the *AbstractTopologyRegistrator* class, which implements the basic and common routines useful in the registration process. The registration process of a domain begins when the application container of the NRPS Adapter (i.e. Tomcat) is started. At startup time, the Adapter application (1) reads the text-based configuration files where the information is placed, (2) gets the endpoints from the NRPS, converts all this information to the proper format (XML messages compliant with the Topology-WS) and (3) sends them to the NSP, which stores the information in the database. From then on, the NSP knows (among other information) the location of the WS endpoints of the Adapter that allow the NSP to perform reservation operations.

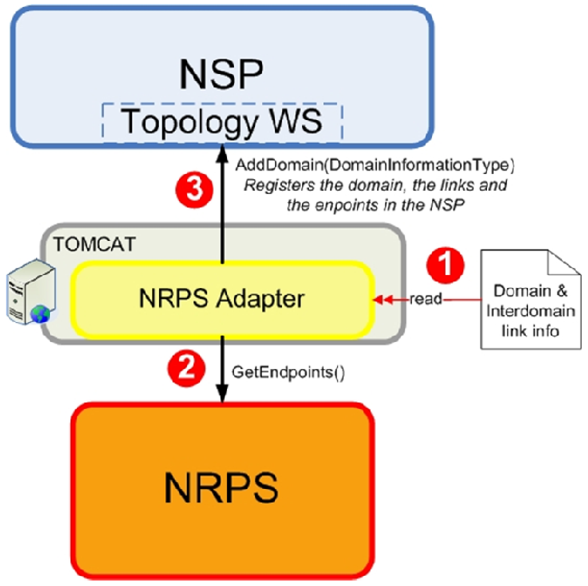

NSP
Topology WS
AddDomain(DomainInformationType)
Registers the domain, the links and the enpoints in the NSP
3
TOMCAT
NRPS Adapter
1
read
Domain & Interdomain link info
2
GetEndpoints()
NRPS

Figure 2.1: Registration process of a domain

In general, an NRPS Adapter reads the information about the domain and the interdomain links from configuration files, and the information about endpoints is requested from the NRPS directly. As it can be seen in the previous figure, the NRPS Adapter pushes all the information through the Topology-WS interface of the NSP (through the *AddDomain(DomainInformationType)* operation, which includes the information about the domain, the links and the endpoints), and uses the specific *GetEndpoints* operation of the NRPS to retrieve the topology.

The information that can be found in the configuration files includes the following:

- Domain name
- Domain description
- Reservation WS EPR of the Adapter (to invoke the reservation operations from the NSP)
- Topology WS EPR of the parent NSP (to register and send the topology information)
- TNA prefix list
- Interdomain links information
- Specific information of the NRPS of the domain (URIs, access information, …)

This information and the one obtained with the *GetEndpoints* operation invocation is enough to register a domain and its topology, as well as to build automatically in the NSP the logical interdomain topology, explained in the next section.

2.1.1.2 *Topology setup*

In this implementation, the topology information of each domain needed by the NSP is distributed between the configuration files of the Adapter and the information contained in the NRPS. The NRPS Adapter can provide (through the GetEndpoints operation of the Reservation-WS) the information about the border endpoints of the domain itself, i.e. the endpoints connected to other Domains, but not the interdomain links information, that must be inserted manually to the configuration files of each Adapter. This is because an NRPS is aware only of its local topology (endpoints and intradomain links), and does not have information about external resources or links, since it is supposed to be an autonomous system by definition. Hence, this interdomain topology information is contained in the configuration file present in each one of the Adapters that control the domains.

The configuration file stores the following information for interdomain links:

- Number of peers. For each peer:
    - Peer name
    - Number of links. For each link:
        - Link name
        - Local source endpoint of the link

This information is sent to the NSP when the Adapter registers. Then the NSP can build the interdomain topology by matching the link names of the neighbouring Domains. Thus, there must be an agreement for the names of the links between the interconnected Domains to make possible this matching in the NSP. The convention can be e.g. the number of the VLAN connecting the Domains (*VLAN937* in the example below).

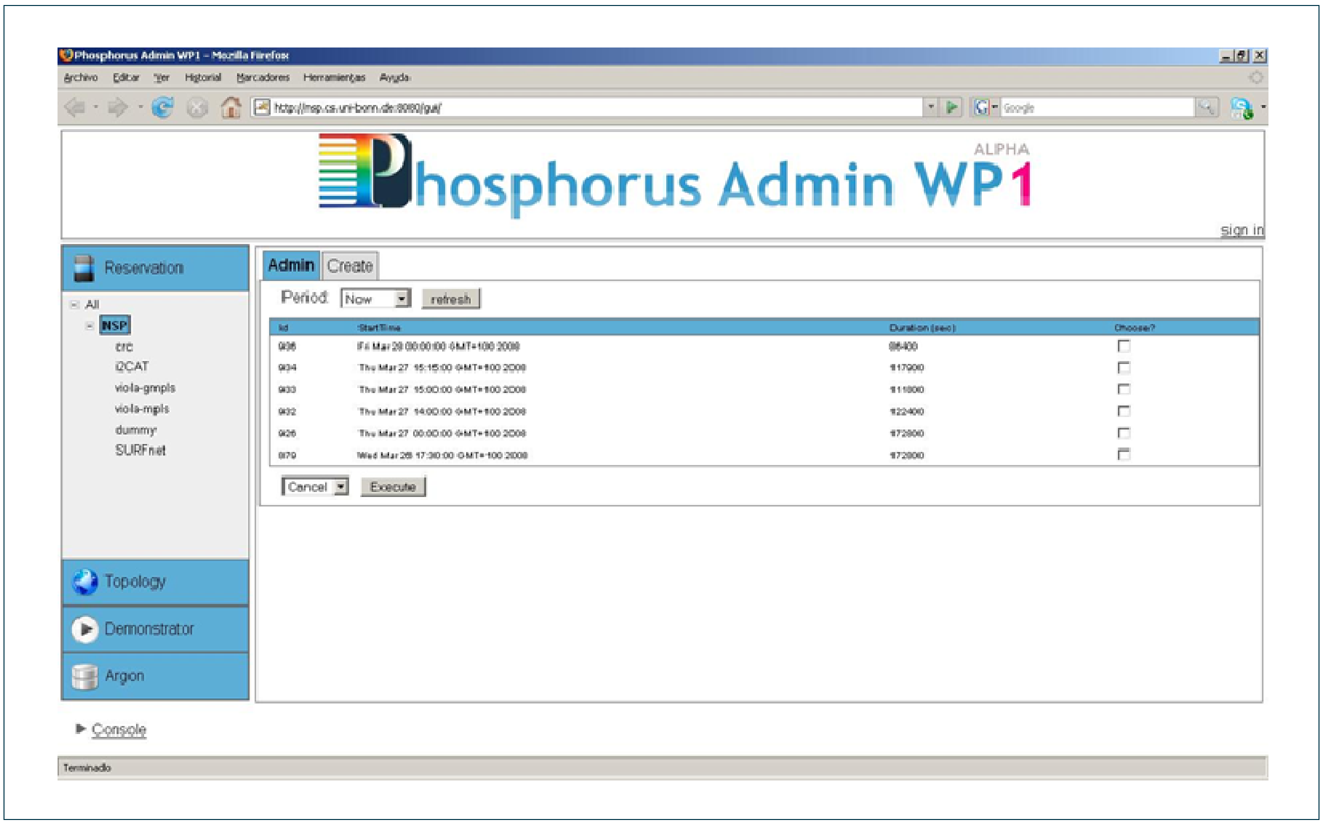
As an example, the following table shows excerpts of the configuration files for i2CAT and VIOLA, where there is a unique link connecting them through the ports 10.3.1.25 (i2CAT) and 10.7.3.1 (VIOLA):

i2CAT		**VIOLA**	
peer0.name = viola	(remote peer)	peer1.name = i2CAT	(remote peer)
peer0.numLinks = 1		peer1.numLinks = 1	
peer0.link0.name = VLAN937		peer1.link0.name = VLAN937	
peer0.link0.srcEP = 10.3.1.25	(local endpoint)	peer1.link0.srcEP = 10.7.3.1	(local endpoint)

Besides using this process, the network topology of the NSP can be managed through the Topology GUI developed for the project (see D1.4 section 4.1). This tool allows a user to connect to the Topology-WS of the NSP and make manually add/edit/delete/query operations for the topology elements (domains, endpoints and links) in a graphical way.

2.1.2 Testbed status retrieval

Once the topology is stored in the NSP and reservations have been created, the status of the testbed can be retrieved in several ways. Besides the tool to manage the topology mentioned in the previous section, there is the administrative GUI that allows creating and cancelling reservations and getting their status. This GUI also shows the domains registered to the NSP and can be used to directly access the Reservation-WS of the domains. For details concerning reservation management with the administrative GUI, see also the following subsections.



Id	StartTime	Duration (sec)	Choose?
936	Fri Mar 28 00:00:00 GMT+100 2008	86400	☐
934	Thu Mar 27 15:15:00 GMT+100 2008	117900	☐
933	Thu Mar 27 15:00:00 GMT+100 2008	111600	☐
932	Thu Mar 27 14:00:00 GMT+100 2008	122400	☐
926	Thu Mar 27 00:00:00 GMT+100 2008	172800	☐
879	Wed Mar 26 17:30:00 GMT+100 2008	172800	☐

Figure 2.2: Administrative GUI showing the reservations

The GUI is based on the Google Web Toolkit API, and is accessible through any Web Browser. It has several sections to manage the different elements of the system. From the Reservation section, reservations can be established, cancelled and their status can be retrieved. The GUI allows showing the reservations of the whole NSP or of any of the Domains. If the reservation ID is selected, the tool sends a *getStatus* message to the correspondent domain and then the status of the reservation is shown.

The Demonstrator section contains an application that uses Google maps that allows the user to create reservations and view the topology (endpoints and interdomain links) over the world map and the status of these links.

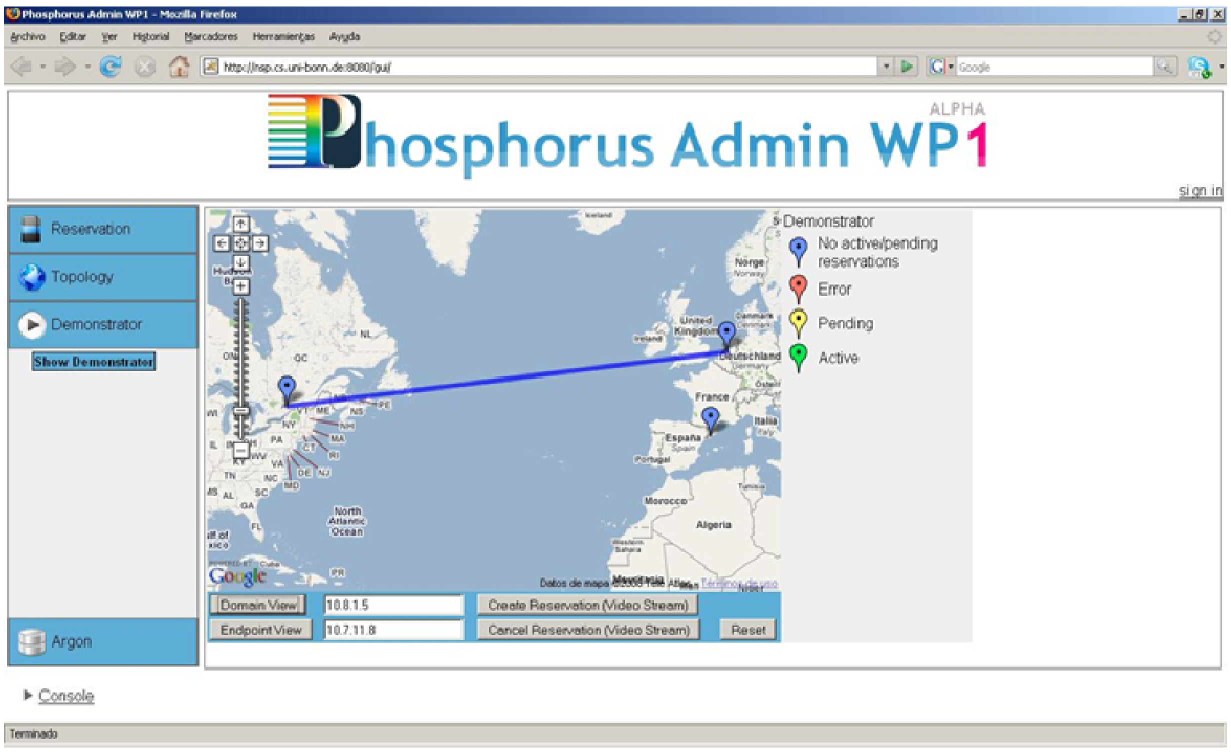

Phosphorus Admin WP1 - Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas Ayuda
http://nsp.cs.uni-bonn.de:8080/gui/
Phosphorus Admin WP1
ALPHA
sign in
Reservation
Topology
Demonstrator
Show Demonstrator
Argon
Demonstrator
No active/pending reservations
Error
Pending
Active
Domain View
Endpoint View
10.8.1.5
10.7.11.8
Create Reservation (Video Stream)
Cancel Reservation (Video Stream)
Reset
Console
Terminado

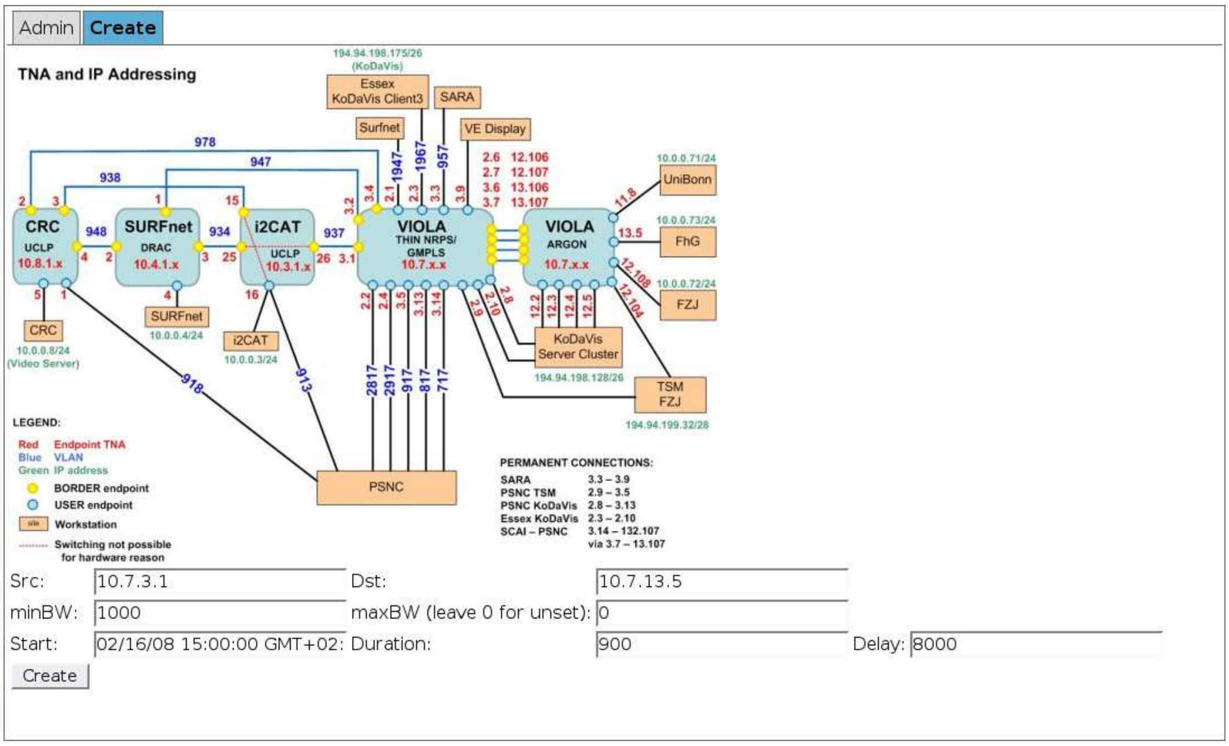
Figure 2.3: Demonstrator GUI of the Administrative tool

2.1.3 Connection establishment

Connections can be set-up through the *CreateReservation* operation of the Reservation-WS. When a *CreateReservation* request is received, the corresponding routine in the *ReservationSetupHandler* class is called.

The requested services are used as input for the *getAvailableServiceList* routine. This routine queries the *PathComputer* for paths for all of the requested connections and splits the single multidomain request to multiple single domain requests, one for each of the involved domains. These requests are handed to the *NRPSManager* that takes care of sending these requests to the NRPSs and collecting the corresponding replies. The *PathComputer* is designed to calculate interdomain paths using the Dijkstra algorithm. When calculating a path, blocking of resources is taken into account. If the requested resources are not available in one or more of the involved domains, they are pruned from the *PathComputer* instance and the *PathComputer* is queried for an alternative path.

This process is repeated until either a suitable path is found, or until so many resources have been pruned that no path is available for one or more connections. In case a path is found consisting of domains that all gave a positive reply to the availability query, the final reservations for all intradomain paths are established by sending the *CreateReservation* messages to each involved NRPS. A global reservation ID is created by the NSP. The NSP keeps a mapping of each local reservation ID assigned by the involved NRPS to the global reservation ID of the NSP which is returned to the user.



PERMANENT CONNECTIONS:

SARA	3.3 – 3.9
PSNC TSM	2.9 – 3.5
PSNC KoDaVis	2.8 – 3.13
Essex KoDaVis	2.3 – 2.10
SCAI – PSNC	3.14 – 132.107 via 3.7 – 13.107

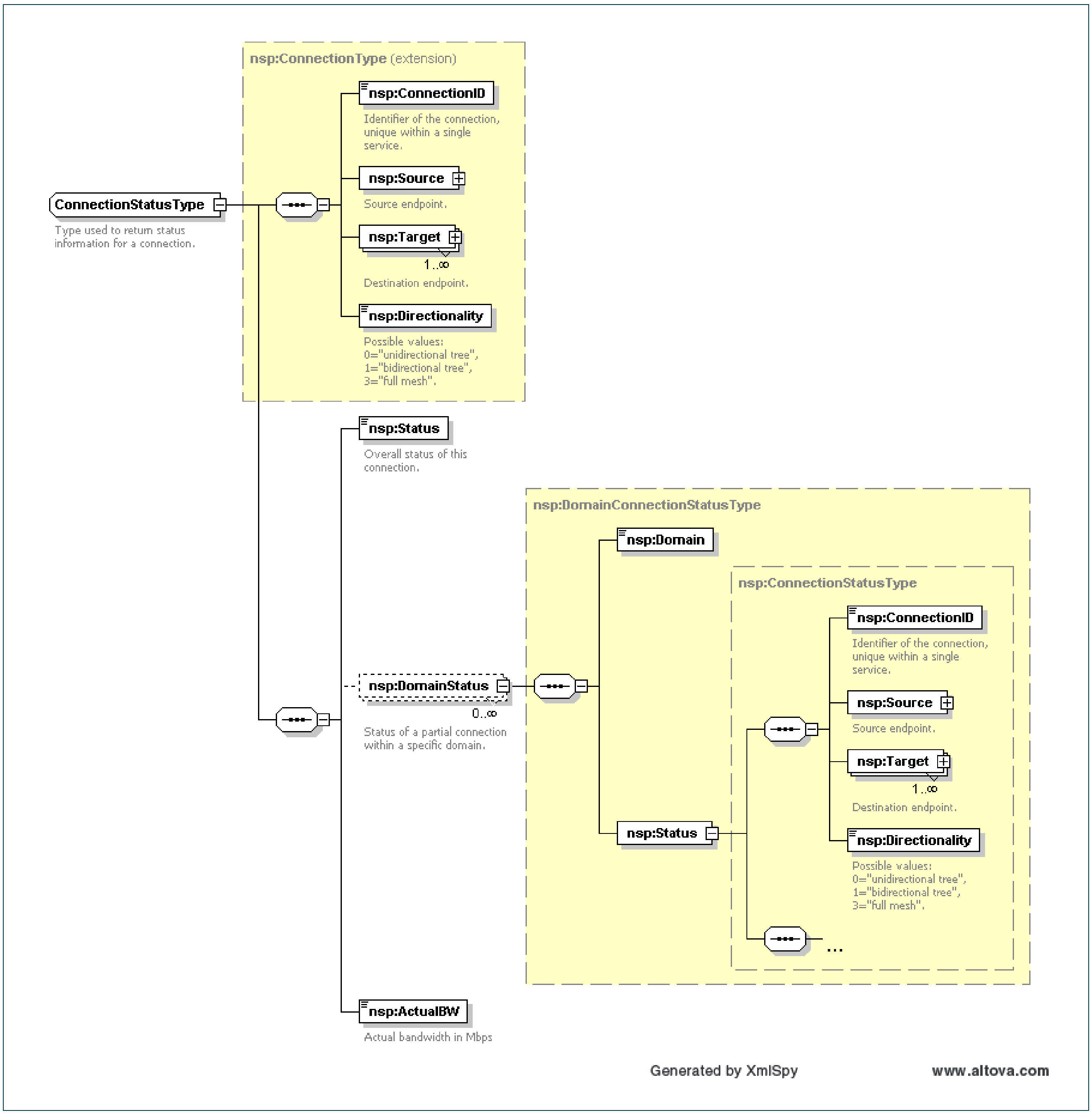
Src:	10.7.3.1	Dst:	10.7.13.5		
minBW:	1000	maxBW (leave 0 for unset):	0		
Start:	02/16/08 15:00:00 GMT+02:	Duration:	900	Delay:	8000

Create

Figure 2.4: Create tab of the management GUI

To trigger the createReservation operation, the management GUI introduced in the previous section is used. After selecting a domain within the Reservation section of the GUI, reservation can be entered into the form on the Create tab (cf. Figure 2.4). After pressing the Create button, a CreateReservationRequest with the according data is sent to the Reservation-WS of the selected domain. In case the request is successful, a window with the reservation ID pops up, else the window contains an error message (e.g. a PathNotFoundFaultException).

2.1.4 Connection status retrieval



nsp:ConnectionType (extension)
nsp:ConnectionID
Identifier of the connection, unique within a single service.
nsp:Source
Source endpoint.
nsp:Target
1..∞
Destination endpoint.
nsp:Directionality
Possible values: 0="unidirectional tree", 1="bidirectional tree", 3="full mesh".
ConnectionStatusType
Type used to return status information for a connection.
nsp:Status
Overall status of this connection.
nsp:DomainConnectionStatusType
nsp:Domain
nsp:ConnectionStatusType
nsp:DomainStatus
0..∞
Status of a partial connection within a specific domain.
nsp:Status
...
nsp:ActualBW
Actual bandwidth in Mbps
Generated by XmlSpy
www.altova.com

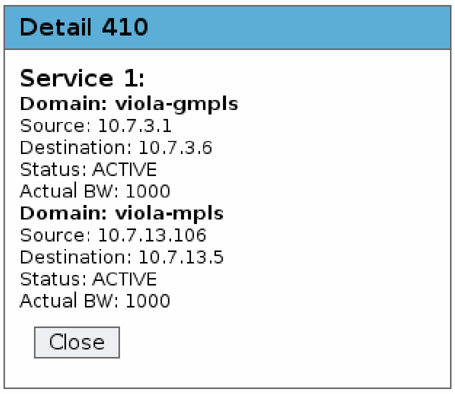
Figure 2.5: Data type used to report the status of a connection

The Reservation-WS offers a *getStatus* operation that allows the querying the status of a reservation. The request for this operation contains the reservation ID, and to restrict the request to a subset of all services contained in the reservation, it may optionally contain a set of service IDs.

The NSP forwards the request to each of the participating domains after adapting the reservation ID assigned to the reservation by the NRPS managing the corresponding domain. The replies received are merged to a reply that is sent back to the user.

Figure 2.5 shows the data type used to return the status of a single connection. This *ConnectionStatusType* is derived from the *ConnectionType* which contains some basic information about the connection, most notably in this context the endpoints that comprise the connection. Additionally, a status reply contains an overall status value for each connection, and optionally one or more *DomainStatus* elements. Each of these elements contains a copy of the *ConnectionStatusType* from the reply of an underlying domain. This discloses the status values of parts of a multidomain connection, which may be of interest to the user when these values differ from on another. If e.g. there is an error in one of the domains, the user can identify which domain causes the error.

Furthermore, since the endpoints of a connection are also contained in the *ConnectionStatusType*, the user can see all border endpoints involved in the connection.


Detail 410

Service 1:
Domain: viola-gmpls
Source: 10.7.3.1
Destination: 10.7.3.6
Status: ACTIVE
Actual BW: 1000
Domain: viola-mpls
Source: 10.7.13.106
Destination: 10.7.13.5
Status: ACTIVE
Actual BW: 1000

Figure 2.6: Window showing the status of an interdomain reservation

To retrieve the status of a connection with the administrative GUI, the user selects a reservation ID from the list of reservation in the Admin tab of a domain in the Reservation section. A GetStatusRequest with this ID is then sent to the domain's Reservation-WS. When the corresponding response is received, a window summarizing the status data pops up (cf. Figure 2.6).

```
<getStatusResponse>
    <ServiceStatus>
        <ServiceID>1</ServiceID>
        <Status>active</Status>
        <DomainStatus>
            <Domain>viola-gmpls</Domain>
            <Status>active</Status>
        </DomainStatus>
        <DomainStatus>
            <Domain>viola-mpls</Domain>
            <Status>active</Status>
        </DomainStatus>
        <Connections>
            <ConnectionID>1</ConnectionID>
            <Source><EndpointId>10.7.3.1</EndpointId></Source>
            <Target><EndpointId>10.7.13.5</EndpointId></Target>
            <Directionality>1</Directionality>
            <Status>active</Status>
            <DomainStatus>
                <Domain>viola-gmpls</Domain>
                <Status>
                    <ConnectionID>1</ConnectionID>
                    <Source><EndpointId>10.7.3.1</EndpointId></Source>
                    <Target><EndpointId>10.7.3.6</EndpointId></Target>
                    <Directionality>1</Directionality>
                    <Status>active</Status>
                    <ActualBW>1000</ActualBW>
                </Status>
            </DomainStatus>
            <DomainStatus>
                <Domain>viola-mpls</Domain>
                <Status>
                    <ConnectionID>1</ConnectionID>
                    <Source><EndpointId>10.7.13.106</EndpointId></Source>
                    <Target><EndpointId>10.7.13.5</EndpointId></Target>
                    <Directionality>1</Directionality>
                    <Status>active</Status>
                    <ActualBW>1000</ActualBW>
                </Status>
            </DomainStatus>
            <ActualBW>0</ActualBW>
        </Connections>
    </ServiceStatus>
</getStatusResponse>
```

Figure 2.7: XML code of GetStatusResponse corresponding to Figure 2.6 (endpoint details omitted)

2.1.5 Connection teardown

The teardown of a running connection is performed by calling the *cancelReservation* operation of the Reservation-WS. Within the NSP, the reservation ID provided as input parameter is mapped to the reservation IDs used locally inside the participating subdomains. A *cancelReservation* with the appropriate ID is then performed within each of these domains.

During demonstration activities, a connection teardown is performed while the data plane is being used by an application (either “ping” or a video stream). It can then be seen that shortly after *cancelReservation* has been called, the connection “freezes”.

To cancel one or more reservations with the administrative GUI, the reservations are checked in the Admin tab of a domain in the Reservation section, Cancel is chosen from the pull-down menu and the Execute button is pressed. Then, CancelReservationRequests are sent to the Reservation-WS of this domain. Afterwards, a window with all results pops up.

2.2 Workflow

2.2.1 Testbed initialization

On startup, the NRPS Adapters in all of the domains register with the central NSP instance by running an *addDomain* or *editDomain* operation of its Topology-WS. The interdomain links are manually added using the graphical topology client, by running a dedicated JUnit test (cf. [JUnit]; such a test is a Java application that takes certain actions and compares the actual results with the expected results) or by manually editing the corresponding table in the NSP database.

2.2.2 Path creation

Path creation is initiated by running the *createReservation* operation of the central NSP instance. The NSP tries to find a feasible multidomain path from a source endpoint to a destination endpoint by querying a path from the *PathComputer* and then checking the feasibility of the path by sending *isAvailable* requests to domains involved in the path. If resources on this path are not available, they are “pruned” from the *PathComputer*. This cycle repeats until either a feasible path is found or until so many resources have been pruned that source and destination endpoint are disconnected.

2.2.3 Path usage

To show that data plane connectivity has been established, command line tools such as ping (simple connectivity check), iperf (available bandwidth check) and traceroute (IP route, hop-by-hop check) are used. The output of the traceroute utility may be used to verify that the connectivity is established on layer 2, i.e. that the hosts are connected via a single IP hop.

During demonstration activities, the connectivity was shown by streaming HD video data between hosts.

2.2.4 Path teardown

The path teardown is a rather simple operation: Upon reception of a *cancelReservation* request, the NRPSManager sends corresponding *cancelReservation* requests to each of the domains involved in the reservation.

3 Test results

3.1 Description of the review meeting demonstration, December 13th 2007 in Poznan

The first reservation made during the demonstration at the review meeting was a connection between the TNAs 10.8.1.5 (connected to a host at CRC) and 10.3.1.16 (connected to a host at the site in Poznan via the i2CAT domain). Prior to the reservation, the interdomain links with the VLANs 934 and 978 were removed (cf. Figure 2.4).

Figure 3.1 shows a protocol of the operations concerning this reservation. The connection spans all four domains CRC, SURFnet, VIOLA, and i2CAT. This is due to the fact that the direct connection between the CRC and i2CAT domains via VLAN 938, which is checked first, is reported to be unavailable by the i2CAT domain that cannot connect the according endpoints for internal reasons. Within the VIOLA domain, the connection is local to the FhG site (third octet of both TNAs is "3"), since both interdomain links, the one to SURFnet as well as the one to i2CAT, terminate there.

The first status requests show the connections to be “active” in all participating domains. However, a “ping” performed between the two hosts showed that there was no connectivity. A subsequent status request (performed at 18:29:56) reveals a failure within the SURFnet domain (reported status was “cancelled_by_system”). The reservation was then cancelled. Now, the other domains report the status of the reservation as “cancelled_by_user”.

```
18:27:25 createReservation[b6067aaf]: 10.8.1.5-10.3.1.16
18:28:03 createReservationResponse[b6067aaf]: 1520 (10.4.1.2-10.4.1.1 /
      10.3.1.26-10.3.1.16 / 10.7.3.2-10.7.3.1 / 10.8.1.5-10.8.1.4)
18:28:05 getStatus[5e6b1c78]: 1520
18:28:11 getStatus[65c0bf21]: 1520
18:28:27 getStatusResponse[5e6b1c78]:
      active:surfnet-testbed,i2CAT,viola-gmpls,CRC
18:28:31 getStatusResponse[65c0bf21]:
      active:surfnet-testbed,i2CAT,viola-gmpls,CRC
18:29:56 getStatus[3bec9b56]: 1520
18:30:12 getStatusResponse[3bec9b56]:
      cancelled_by_system:surfnet-testbed / active:i2CAT,viola-gmpls,CRC
18:30:57 cancelReservation[34115b7c]: 1520
18:31:25 cancelReservationResponse[34115b7c]: true
18:31:31 getStatus[d69be1a4]: 1520
18:31:44 getStatusResponse[d69be1a4]: cancelled_by_system:surfnet-testbed /
      cancelled_by_user:i2CAT,viola-gmpls,CRC
```

Figure 3.1: Poznan-Demo, Reservation #1520

As a result of the failure of the SURFnet domain, the interdomain link table in the NSP database was modified: VLAN 978 was added to the interdomain topology and VLAN 948 was removed. Now, the shortest available connection connects the CRC and the i2CAT domains directly via the VIOLA GMPLS domain (cf. Figure 3.2). Within the VIOLA domain, the connection crosses the FhG and the Uni Bonn sites, since the CRC connection (which physically uses the Uni Bonn – PSNC link) terminates at Uni Bonn.

The connections are reported to be “active” by all domains, and this time, connectivity was shown by running “ping” between the two hosts. A “traceroute” reveals that the hosts are connect on OSI layer 2 on a single IP subnetwork. A live HD video is streamed by the CRC team in Canada to the venue in Poznan.

```
18:32:26 createReservation[7a219579]: 10.8.1.5-10.3.1.16
18:32:55 createReservationResponse[7a219579]: 1521
      (10.3.1.26-10.3.1.16 / 10.7.1.2-10.7.3.1 / 10.8.1.5-10.8.1.2)
18:32:56 getStatus[247a526a]: 1520
18:33:12 getStatus[bee506e7]: 1521
18:33:12 getStatusResponse[247a526a]: cancelled_by_system:surfnet-testbed /
      cancelled_by_user:i2CAT,viola-gmpls,CRC
18:33:12 getStatus[fbaa3e45]: 1521
18:33:33 getStatusResponse[fbaa3e45]: active:i2CAT,viola-gmpls,CRC
18:33:33 getStatusResponse[bee506e7]: active:i2CAT,viola-gmpls,CRC
18:40:46 cancelReservation[9c96440b]: 1521
18:41:13 cancelReservationResponse[9c96440b]: true
18:41:14 getStatus[6d89df4d]: 1521
18:41:16 getStatus[906f1d27]: 1521
18:41:34 getStatusResponse[6d89df4d]:
      cancelled_by_user:i2CAT,viola-gmpls,CRC
18:41:35 getStatusResponse[906f1d27]:
      cancelled_by_user:i2CAT,viola-gmpls,CRC
```

Figure 3.2: Poznan-Demo, Reservation #1521

The second part of the review demonstration was performed by work package 3. The meta-scheduler accessed the central NSP instance through its northbound interface to make the reservations. The requested connections spanned endpoints in the VIOLA GMPLS domain (to which hosts located at the venue of the review were connected) and the VIOLA MPLS domain (to which hosts located in Germany were connected). The middleware was extended to send getStatus requests to the NSP every 15 seconds to become aware of changes in the connections' status. Figure 3.3 shows the first two reservations made by WP3 software during the review meeting demonstration.

```
18:49:39 createReservation[7065eb5a]: 10.7.1.5-10.7.12.3
18:49:46 createReservationResponse[7065eb5a]: 1522
      (10.7.1.5-10.7.3.6 / 10.7.13.106-10.7.12.3)
18:49:46 getStatus[9a71425f]: 1522
18:49:47 getStatusResponse[9a71425f]: active:viola-gmpls /
      setup_in_progress:viola-mpls
18:50:02 getStatus[aabf0f00]: 1522
18:50:05 getStatusResponse[aabf0f00]: active:viola-gmpls /
      setup_in_progress:viola-mpls
18:50:20 getStatus[106b518b]: 1522
18:50:21 getStatusResponse[106b518b]: active:viola-gmpls /
      setup_in_progress:viola-mpls
18:50:36 getStatus[a3c3d266]: 1522
18:50:38 getStatusResponse[a3c3d266]: active:viola-gmpls,viola-mpls
18:50:53 getStatus[2759b1e0]: 1522
18:50:54 getStatusResponse[2759b1e0]: active:viola-gmpls,viola-mpls
18:51:09 getStatus[e79a5afe]: 1522
18:51:11 getStatusResponse[e79a5afe]: active:viola-gmpls,viola-mpls
[...]
18:53:38 getStatus[2d182ad0]: 1522
18:53:39 getStatusResponse[2d182ad0]: active:viola-gmpls,viola-mpls
18:53:55 getStatus[99402a2d]: 1522
18:53:56 createReservation[a5ff8659]: 10.7.1.3-10.7.12.5
18:53:56 getStatusResponse[99402a2d]: active:viola-gmpls,viola-mpls
18:54:04 createReservationResponse[a5ff8659]: 1523
      (10.7.1.3-10.7.1.6 / 10.7.11.106-10.7.12.5)
18:54:04 getStatus[df8bc602]: 1523
18:54:06 getStatusResponse[df8bc602]: active:viola-gmpls /
      setup_in_progress:viola-mpls
18:54:11 getStatus[ce5665f6]: 1522
18:54:13 getStatusResponse[ce5665f6]: active:viola-gmpls,viola-mpls
18:54:21 getStatus[c4cc281c]: 1523
18:54:22 getStatusResponse[c4cc281c]: active:viola-gmpls /
      setup_in_progress:viola-mpls
18:54:28 getStatus[e5ac364a]: 1522
18:54:29 getStatusResponse[e5ac364a]: active:viola-gmpls,viola-mpls
18:54:37 getStatus[052fee66]: 1523
18:54:39 getStatusResponse[052fee66]: active:viola-gmpls,viola-mpls
```

Figure 3.3: Poznan-Demo, Reservations #1522 and #1523 made by the middleware

3.2 Description of tests related to the ONDM'08 demonstration

Due to some hardware failures prior to the demonstration, only a reduced interdomain topology was available. At i2cat, the two endpoints 10.7.3.16 and 10.7.3.26 were not available, and the decision was made to connect the i2cat host to 10.7.3.15 (the new user endpoint) and 10.7.3.25 to VLAN 937 leading to the VIOLA domain.

Furthermore, as a consequence of a technical issue in the SURFnet domain, DRAC could only connect endpoints 10.4.1.1 with 10.4.1.3 and 10.4.1.2 with 10.4.1.4.

With this configuration, connectivity tests were made between all host pairs with the following TNAs (cf. Figure 2.4):

- 10.3.1.15 (i2CAT)
- 10.4.1.4 (SURFnet)
- 10.7.12.108 (VIOLA MPLS)
- 10.8.1.5 (CRC)

The test results are described in the following subsections.

3.2.1 I2CAT – SURFnet

The first path for this connection calculated by the *PathComputer* contains only the transit domain viola-gmpls:

10.3.1.15-10.3.1.25 / 10.7.3.1-10.7.3.2 / 10.4.1.1-10.4.1.4.

Due to the technical issue at SURFnet mentioned above, 10.4.1.1-10.4.1.4 is not available though, and the DRAC Adapter at SURFnet replies to the availability request for this connection with PATH_NOT_AVAILABLE. Therefore, this path is pruned from the path computer's internal topology, and the next path it returns also includes CRC as transit domain:

10.3.1.15-10.3.1.25 / 10.7.3.1-10.7.3.4 / 10.8.1.2-10.8.1.4 / 10.4.1.2-10.4.1.4.

Each of the intradomain connections of this path is reported to be available by the corresponding NRPS Adapters and the reservation is successfully established.

3.2.2 I2CAT – VIOLA MPLS

The i2CAT – VIOLA connection is setup along the path with the least domains, i.e. with the VIOLA GMPLS domain as transit domain.

10.3.1.15-10.3.1.25 / 10.7.3.1-10.7.3.7 / 10.7.13.107-10.7.12.108

3.2.3 I2CAT – CRC

The same holds for the i2CAT – CRC connection, which also is possible through the VIOLA GMPLS domain.

10.3.1.15-10.3.1.25 / 10.7.3.1-10.7.3.4 / 10.8.1.2-10.8.1.5

3.2.4 SURFnet – VIOLA MPLS

The shortest path between SURFnet and the VIOLA MPLS domain

10.4.1.4-10.4.1.1 / 10.7.3.2-10.7.3.7 / 10.7.13.107-10.7.12.108

is not available for the same reason as already described for the i2CAT – SURFnet connection in section 3.2.1. Therefore, the connection is set up via CRC:

10.4.1.4-10.4.1.2 / 10.8.1.4-10.8.1.2 / 10.7.3.4-10.7.3.6 / 10.7.13.106-10.7.12.108

It can also be seen here that the *PathComputer* does not actually resort to partial paths whose availabilities have already been checked. In this example, a different link between the VIOLA MPLS and GMPLS domains has been chosen for the second path, although the link in first path was confirmed to be available.

3.2.5 SURFnet – CRC

Between SURFnet and CRC, the direct connection is available:

10.4.1.4-10.4.1.2 / 10.8.1.4-10.8.1.5

3.2.6 VIOLA MPLS – CRC

The shortest connection between the VIOLA MPLS domain and CRC traverses the VIOLA GMPLS domain and is available:

10.7.12.108-10.7.13.107 / 10.7.3.7-10.7.3.4 / 10.8.1.2-10.8.1.5

3.3 Single domain tests

This section describes a JUnit test suite, each test involving only a single NRPS Adapter. A test run consists of the following workflow:

1. Connections setup between two endpoints located in the same domain.

2. After a short delay, query the connection's status (and expect it to be *active*).

3. Try to create another reservation between the same two endpoints (and expect this to fail). The availability reply received from the NRPS Adapter should carry the availability code *endpoint_not_available* and a list of blocked endpoints containing both the source and destination.of the reservation request.

4. Cancel the connection.

5. After a short delay, query the connection's status (and expect it to be *cancelled_by_user*).

The detailed results of these tests are not displayed here as they were successful for all domains. Merely the availability reports for another connection between the two endpoints received after the first connection has successfully been set up are slightly different for the different systems. None of the received availability reports are actually wrong. In larger scenarios, it is merely more efficient to report that specific endpoints are blocked instead of reporting the path between them to be unavailable.

3.3.1 I2CAT

The UCLP Adapter at i2CAT reports *endpoint_not_available* in its availability response for the second reservation request for the endpoints already blocked by the first reservation. However, the list of blocked endpoints is empty, the response is therefore treated like a *path_not_available*.

3.3.2 SURFnet

The DRAC Adapter at SURFnet reports *path_not_available* in its availability response for the second reservation request for the endpoints already blocked by the first reservation.

3.3.3 VIOLA MPLS

The ARGON Adapter at the VIOLA MPLS domain reports *endpoint_not_available* in its availability response for the second reservation request for the endpoints already blocked by the first reservation. The list of blocked endpoints contains both endpoints.

3.3.4 VIOLA GMPLS

The Thin NRPS at the VIOLA GMPLS domain reports *endpoint_not_available* in its availability response for the second reservation request for the endpoints already blocked by the first reservation. The list of blocked endpoints contains both endpoints.

3.3.5 CRC

The UCLP Adapter at CRC reports *endpoint_not_available* in its availability response for the second reservation request for the endpoints already blocked by the first reservation. However, the list of blocked endpoints is empty, the response is therefore treated like a *path_not_available*.

3.4 Path finding tests

In this test, the dynamic allocation to available resources is verified. The following test workflow is processed:

1. Create an immediate reservation with a duration of 5 minutes between the CRC host at 10.8.1.5 and the VIOLA host at 10.7.12.108. The direct connection between CRC and VIOLA (across VLAN 978) should be taken.

2. Block VLAN 978 by creating an advance reservation between the VIOLA-GMPLS endpoints 10.7.3.4 and 10.7.12.107, starting in 6 minutes, with a duration of 1 minute.

3. Create a reservation between the CRC host at 10.8.1.5 and the VIOLA host at 10.7.12.108 like in step 1, however starting at the same time as the reservation made in step 2. This time, a connection via SURFnet should be taken.

4. Create a reservation between the i2CAT host at 10.3.1.16 and the VIOLA host at 10.7.13.5, starting at the same time as the reservation made in step 3. A different link between the VIOLA-MPLS and VIOLA-GMPLS domains should be chosen.

This test workflow creates four reservations, one for each of the four steps. The interdomain paths taken are the following:

1. 10.8.1.5-10.8.1.2 / 10.7.3.4-10.7.3.7 / 10.7.13.107-10.7.12.108

2. 10.7.3.4-10.7.12.107

3. 10.8.1.5-10.8.1.4 / 10.4.1.2-10.4.1.1 / 10.7.3.2-10.7.3.7 / 10.7.13.107-10.7.12.108

4. 10.3.1.16-10.3.1.26 / 10.7.3.1-10.7.2.6 / 10.7.12.106-10.7.13.5

The results show that interdomain paths are dynamically chosen by the *PathComputer* depending on the availability of resources, which is managed by the underlying NRPSs.

The results also show that paths are chosen optimally only with reference to the number of interdomain links traversed: For the last connection (i2CAT-VIOLA), an interdomain link in Sankt Augustin is chosen. This means that in the VIOLA-GMPLS domain a connection is made from Jülich to Sankt Augustin, and in the VIOLA-MPLS domain from Sankt Augustin back to Jülich (cf. Section 1.1.3.1), although the second interdomain link in Jülich was also available. Using this link would have saved resources on the links between VIOLA-GMPLS and VIOLA-MPLS for future connections.

4 Towards G^{2}MPLS integration

The philosophy of G^{2}MPLS is to follow a completely distributed approach for resource provisioning, while the philosophy of the Network Service Plane developed in WP1 is strongly influenced by the centralized approach of Networks Resource Provisioning Systems. Therefore, it is not trivial to implement an interworking between G^{2}MPLS and the Network Service Plane.

In discussions between WP1 and WP2, two possible options have been identified to achieve the interworking between G^{2}MPLS and the NSP. The simple client-server approach is to integrate G^{2}MPLS domains into a NSP architecture in the same way as GMPLS domains, by using a “Thin NRPS Adapter”. This Adapter would actually require even less functionality than for plain GMPLS domains, since the resource usage is tracked by G^{2}MPLS and therefore must not be tracked by the Thin NRPS Adapter.

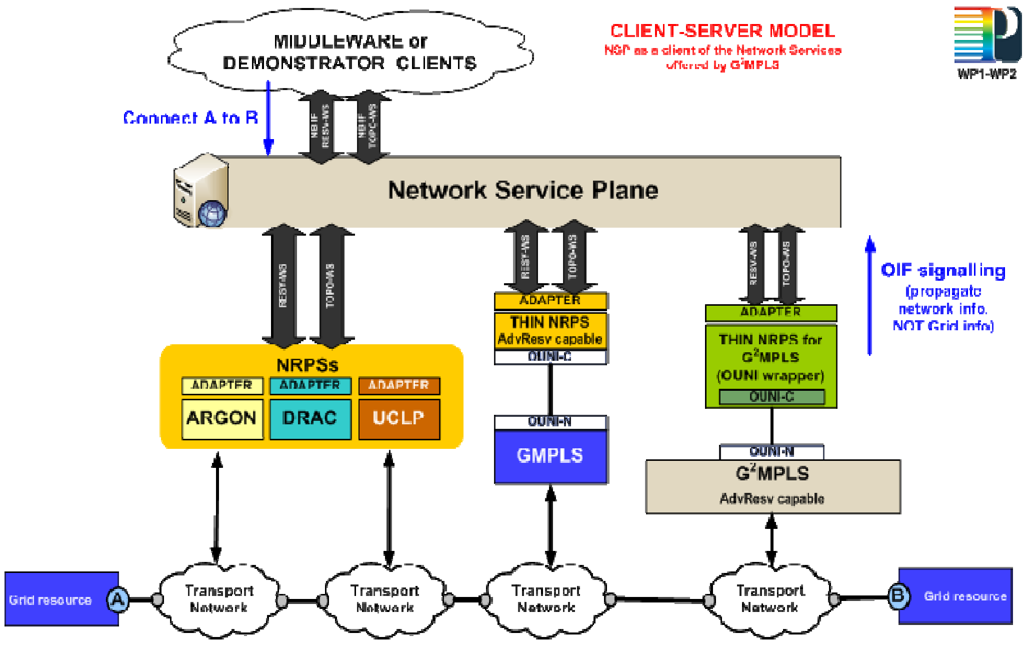


Figure 4.1: Client-server communication between the NSP and G^{2}MPLS.

This simple approach has the disadvantage that it does not allow G^{2}MPLS domains to be aware of the other domains and to direct reservation requests to these domains; it would only be able to forward reservation requests with unknown endpoints to its parent NSP instance.

A more flexible and challenging solution is to implement a peering model. According to this model, NSP domains and G^{2}MPLS domains exchange domain and interdomain link information such that each domain is able to internally generate a representation of the interdomain topology, allowing it to make path computations and to send requests directly to other domains. No central NSP instance would be necessary in this model. The information has to be translated by a gateway that communicates using a Topology-WS interface on the one side and a G^{2}MPLS E-NNI interface on the other side. A G^{2}MPLS E-NNI gateway for interworking between G^{2}MPLS and other systems is already being planned by WP2, so a reasonable approach would be to implement the conversion within this gateway as joint WP1/WP2 effort.

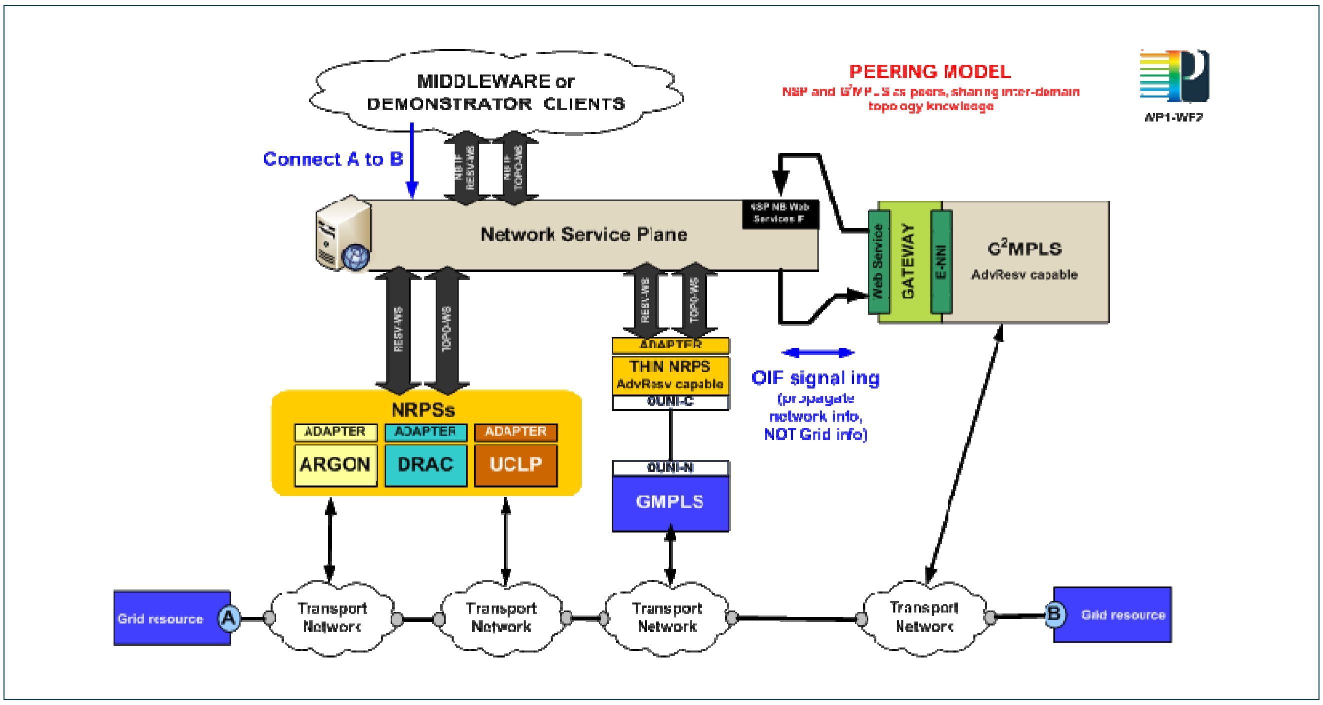


Figure 4.2: Peer-to-peer communication between the NSP and G^{2}MPLS.

Regardless of the model used, it must be clear that the NSP, as the network service provider entity below the middleware, is not designed to be aware of Grid resources. Therefore, G^{2}MPLS entities can only request network resources, not Grid resources.

In discussions between WP1 and WP2, it has been concluded that the peering model is feasible, as the topology information exchanged between G^{2}MPLS domains is not too different from the topology information exchanged between NSP domains. However, there are several differences in some details, and solutions for unifying these differences remain to be elaborated.

4.1 Topology exchange

G^{2}MPLS (cf. [Phosphorus-D2.7]) uses E-NNI routing for disseminating information about Routing Control Domains (RCD). The main purpose of the G^{2}MPLS E-NNI routing is the interdomain information flooding of local and learned Grid and network resources (see below). The NSP in turn does the same (albeit only for network resources) through the Topology-WS interface. Hence, a common topology exchange mechanism has to be defined in order to disseminate this information between these two types of domains.

For G^{2}MPLS, the main entities taking part in the interdomain routing exchange are interdomain Routing Controllers (G.eNNI-RC). The G.eNNI-RC represents a Routing Control Domain (RCD). Every RCD has one and only one G.eNNI-RC responsible for the external view of the domain. The G.eNNI-RC establishes routing adjacencies with every G.eNNI-RC placed in the neighbouring RCDs. Routing adjacencies are used to exchange G.E-NNI routing messages. The G.eNNI-RC gathers resource information from the local domain and floods the information to the adjacent G.eNNI-RCs. Additionally, the G.eNNI-RC learns the information from the adjacent domain and it propagates the information on the other adjacencies and also in the local domain. Taking this into account, at a first glance, the interoperability between G^{2}MPLS and the NSP should be between the G.eNNI-RC and the Topology-WS. This would lead the partners to implement some kind of adaptation layer/s or gateway/s between the two types of domains to allow the exchange of information (natively, WS-XML for the NSP and G.OSPF-TE for G^{2}MPLS).

4.2 Signalling

G^{2}MPLS E-NNI signalling is conceived to cope with:

- Grid job/service requests translated in setup of GNS (Grid Network Service) calls
- support of advance reservations
- support of implicit network destinations for Calls related to a Grid service (destination being an anycast for e.g. an amount of CPU power or storage capacity)

The G.E-NNI signalling interface exists between two signalling control domains and supports the procedures of the ASON architecture, extended with the GNS transaction related information. G.E-NNI signalling must be compatible with call/connection control originating from or destined to a G.OUNI complaint interface. The signalling of the NSP (only for network reservation services – not Grid) is performed through the Reservation-WS interface. As in the topology case, a common approach for interoperation must be discussed.

The signalling messages of the NSP are defined with XML schema files manageable by the Reservation-WS and consequently, they have XML format. The selected signalling protocol for G^{2}MPLS operations across the G.E-NNI interface is RSVP, with its standard extensions contributed by IETF for the GMPLS part and by OIF for the E-NNI parts. As can be seen, the two methods for network to network communication are quite different. The first approach for their interoperability would be an adaptation/translation process of the messages between the two types of domain through an adaptation layer or gateway.

The following image depicts a possible scenario for the interoperation between the NSP and G^{2}MPLS. In this solution, a gateway would be implemented for WS intercommunication. This gateway would make the adaptation/translation of the routing/signalling messages of the G^{2}MPLS E-NNI to the standard WS format of the NSP and the NRPS Adapters. The gateway might use the AbstractRegistrator already implemented by WP1 for the NRPS Adapters in order to automatically register with an NSP and send the topology of the associated domain. In its turn, the NSP should implement some flooding mechanism to send the own topology information to the gateway.

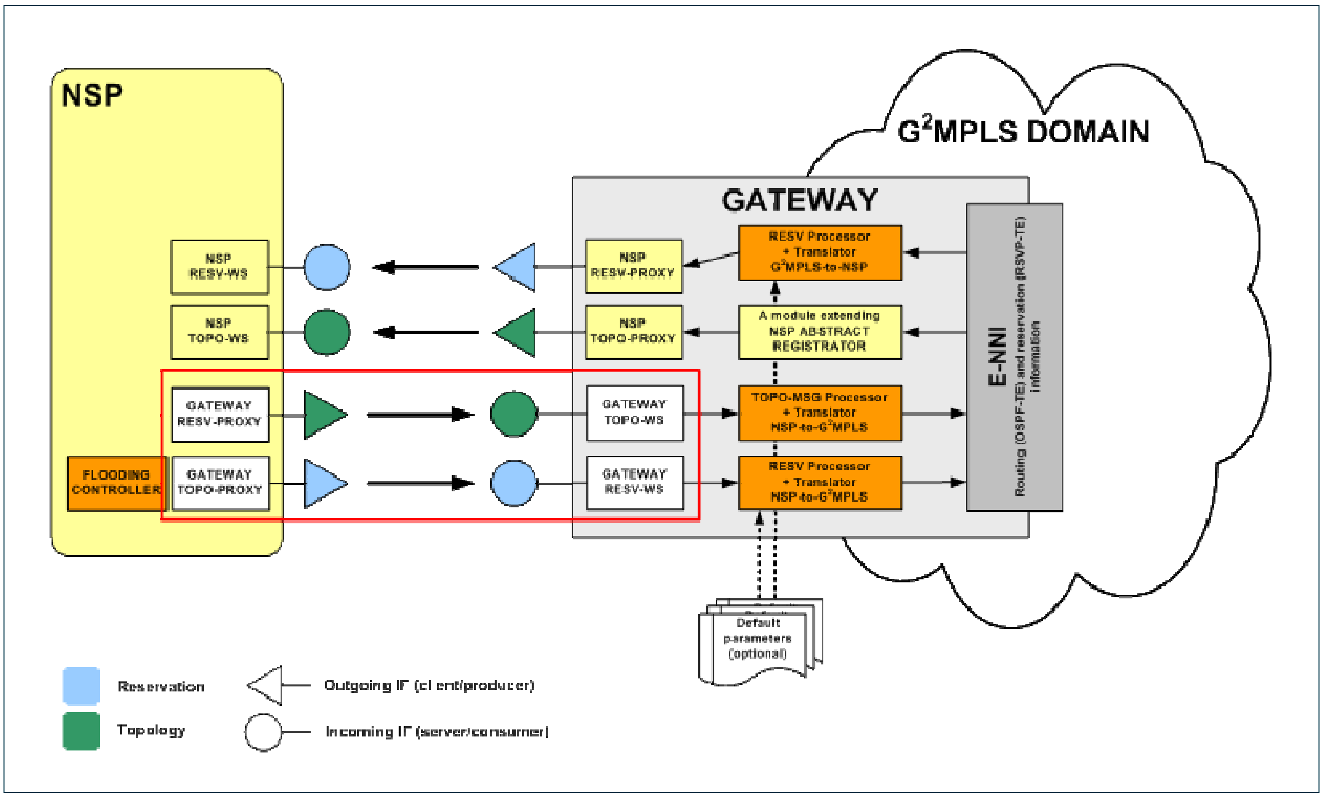

NSP
G²MPLS DOMAIN
GATEWAY
NSP RESV-WS
NSP TOPO-WS
GATEWAY RESV-PROXY
GATEWAY TOPO-PROXY
FLOODING CONTROLLER
NSP RESV-PROXY
NSP TOPO-PROXY
GATEWAY TOPO-WS
GATEWAY RESV-WS
RESV Processor + Translator G²MPLS-to-NSP
A module extending NSP ABSTRACT REGISTRATOR
TOPO-MSG Processor + Translator NSP-to-G²MPLS
RESV Processor + Translator NSP-to-G²MPLS
E-NNI
Routing (OSPF-TE) and reservation (RSVP-TE) information
Default parameters (optional)
Reservation
Topology
Outgoing IF (client/producer)
Incoming IF (server/consumer)

Figure 4.3: Schema of possible interoperation between the NSP and G^{2}MPLS.

4.3 Negotiation and resource scheduling

When implementing a peering model for interoperation between NSP and G^{2}MPLS, the difference between these systems in the way interdomain routing is achieved has to be taken into account. This difference stems from the different underlying philosophies already noted in the introductory remarks: The NSP design is strongly influenced by the centralized approach followed by traditional NRPSs. Therefore, the domain receiving a reservation request will query every other domain for possible intradomain connections within this domain. G^{2}MPLS in contrast implements a chain model, i.e. domains will contact only their neighbour domains querying for partial interdomain reservations. These two approaches are sketched in Figure 4.4.

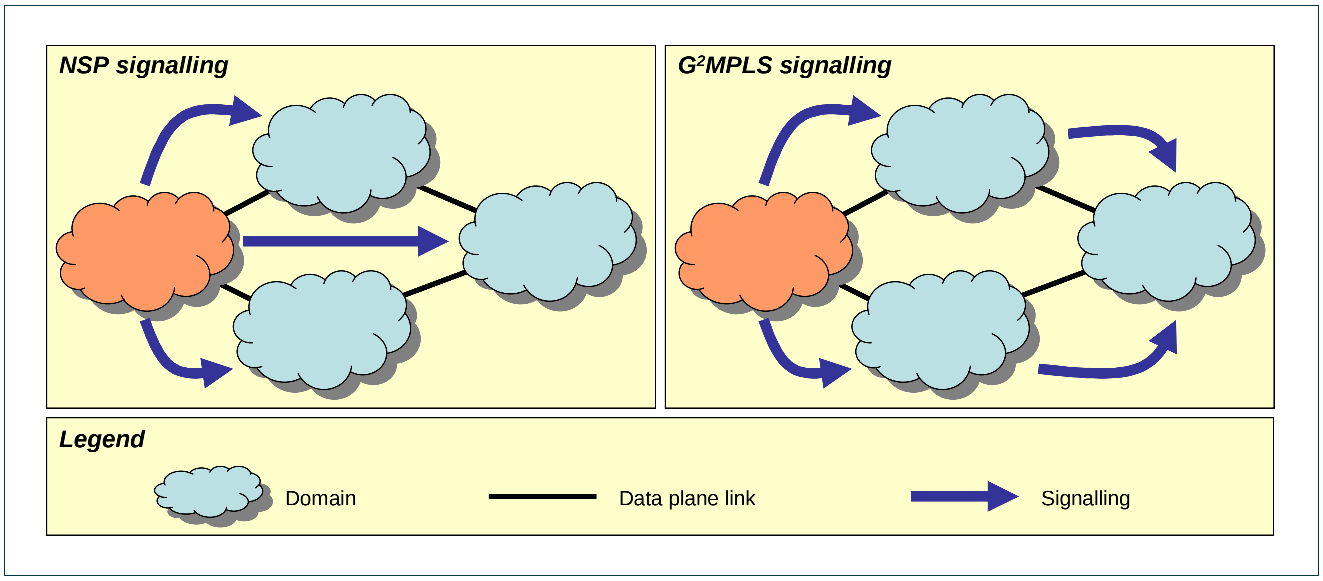


Figure 4.4: Schemes of possible interoperation between the NSP and G^{2}MPLS.

To allow NSP and G^{2}MPLS to interoperate in negotiation and resource scheduling given these different approaches, two possibilities (and combinations thereof), their advantages and disadvantages should be further explored:

1. A gateway between an NSP and a G^{2}MPLS domain will have to take care to mediate between the different approaches to interdomain routing.

2. Only specific interdomain architectures are allowed (i.e. a single NSP superdomain that subsumes all NSP domains peering with one or more G^{2}MPLS domains).

4.4 AAI issues

4.4.1 Level of Security

Security in this context can be located on the network, transport, and message level (cf. Figure 4.5). Within the WP1 testbed, the control plane communication is secured by using **network level security** (NLS). Each involved system is part of a virtual private network (VPN) that was created by using the free software tinc[tinc] (cf. Section 1.3). Communication from other locations (e.g. for the user GUI) is allowed for a reduced set of source IP address ranges only. The G^{2}MPLS could easily be added to the VPN or to the allowed address range.

Since the security within the NSP is based on NLS, no **transport level security** (TLS) mechanisms are implemented at the moment. In order to communicate with other systems (e.g. such as the Internet2 IDC) it might be necessary to provide security for this level using SSL. This could be integrated into the different systems within the NSP domain as well as in the gateway to the G^{2}MPLS without any major problems.

The **message level security** (MLS) mechanisms are the main focus of the Phosphorus WP4 AAI integration. This is why the discussion has to be distinguished between the authentication (AuthN) and authorization (AuthZ) phase.

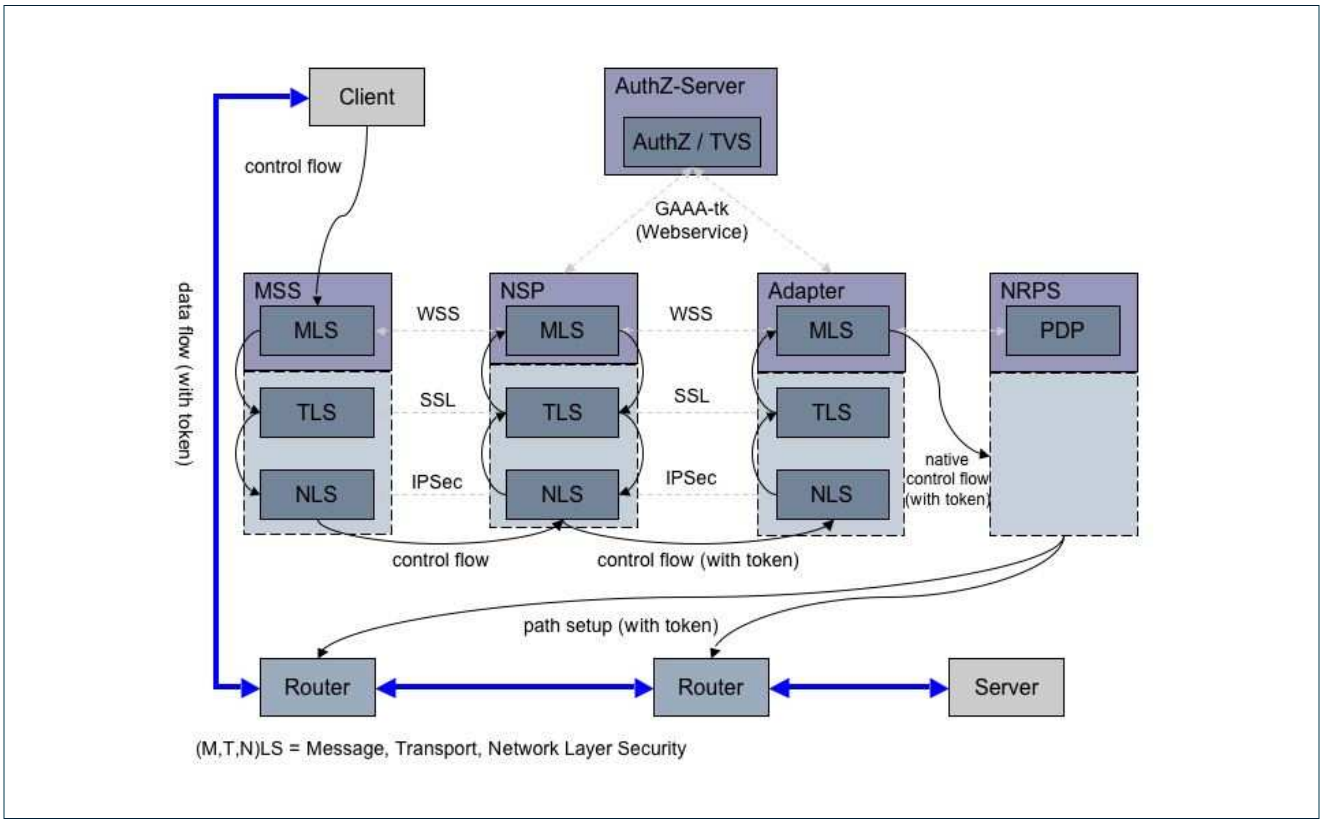


Figure 4.5: Overview of the security levels.

4.4.2 Authentication

The NSP contains (within each Adapter) a central module that is used for **AuthN** aspects. It is used to authenticate/decrypt all incoming and to sign/encrypt all outgoing traffic. This service is based on the OASIS Web services Security standard [WSS]. Besides procedures to sign and to encrypt SOAP messages the standard includes options to attach security credentials like username/password, X.509 certificates or tokens. Since WP3 is using Apache Rampart [rampart] for the very same functionalities, this well known software module might also be used within the NSP to avoid incompatibilities with G^{2}MPLS.

Figure 4.6 depicts a sequence of interactions needed for the authentication flow between the interacting systems. The following sequence description is simplified and reduced to a single MSS-NSP-G^{2}MPLS communication for AuthN: (1) An MSS client sends a request to the Meta-Scheduling Service (MSS) with local user credentials. (2) The MSS authenticates and authorizes the user and the request locally. In case the request is authorized successfully, the scheduler maps the user credentials to accordant global attributes, adds these to the request to the NSP and signs the message with its private key. (3) The message then is sent to the NSP on behalf of the client. Since the public key of the MSS is trusted within the NSP, the message is accepted in the next step. The signature of the valid incoming request will be removed and the request may be split into several new requests. (4) The outgoing messages to the G^{2}MPLS gateway are signed by the NSP. All authorization related information that may be added by the MSS is forwarded without any modification. (5) In the expected case that the G^{2}MPLS gateway trusts the NSP key all authorization information (e.g. global attributes, tickets) and the request is forwarded to the specific NRPS. (6) A complex authorization process has to be implemented in the G^{2}MPLS gateway or the underlying systems itself.

This way, the service plane acts as a transparent broker between the MSS and the G^{2}MPLS gateway. It is self-evident that this message level security flow is also applied for the corresponding response messages. Additionally, this architecture could be used to encrypt the whole message flow.

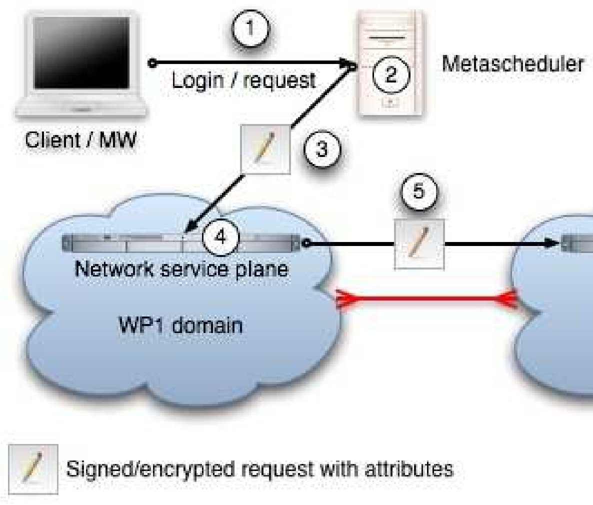


Figure 4.6: Authenticated message flow between MSS, NSP and G^{2}MPLS

4.4.3 Authorization

For the request **AuthZ** process WP1 is integrating the WP4s GAAA-tk (over Web service) into the service plane and will use it for all AuthZ related issues within the communication flow. Since G^{2}MPLS is also planning to integrate the GAAA-tk, incompatible solutions should be avoided. As depicted in Figure 4.6 the MSS acts as an Attribute Authority (AA) that serves the role of a trusted entity for the service plane that mediates requests for holders of digital credentials. It must have privileged access to the local authentication domain database that holds information (identity attributes) about the credential holders. The MSS operates on rulesets defining what attributes can be attached to the request and under what circumstances. The service plane itself uses the GAAA-tk to authorize the request with its attached attributes. Then the NSP forwards the request with the user credentials (attributes) that are contained in the incoming message from the middleware to the involved G^{2}MPLS gateway and vice versa. The same is then done within the G^{2}MPLS domain.

It is assumed that each domain has its own policy and attribute database. The G^{2}MPLS gateway may map the global attributes to local ones. In the case that global and local attributes are identical, this mapping reduces to the identity function.

Within WP4, the path creation and path administration (and additionally its usage) is treated in different ways. This is why the two subsequent chapters describe the desired AuthN and AuthZ workflow in a more detailed way.

4.4.3.1 *Path creation*

In **Figure 4.7** a generalized AuthN/AuthZ workflow for a path creation process is depicted. The different steps can shortly be described as follows: (1) The client – in this case for example the MSS – creates [action| a reservation from A to B for a specific time frame [resource|. The request is signed by the clients certificate [credentials] and encrypted with the NSP/IDC/G^{2}MPLS-GWs public key. (2) The NSP/IDC/G^{2}MPLS-GW validates the signature of the incoming message by comparing it with the pre-installed public keys. (3) After the successful authentication parts of the request will be sent to an AuthZ server by using the GAAA-tk. This message includes a global reservation identifier (GRI) created by the NSP/IDC/G^{2}MPLS-GW, the action, resources and credentials. The AuthZ server in return will send back a token. (4) Now the NSP/IDC/G^{2}MPLS-GW creates a new reservation request for the involved NRPSs/NSPs/IDCs/G^{2}MPLS-GWs including the token and the GRI. (5)(6) The next system now runs the AuthN and AuthZ process again for the incoming request. (7) Thereafter the underlying network elements (NEs) are configured (in case of an immediate reservation) and a local reservation ID (LRI) is sent back in step (8). (9) Finally the client receives the NSPs/IDCs/G^{2}MPLS-GWs LRI, the GRI and the token for the reservation.

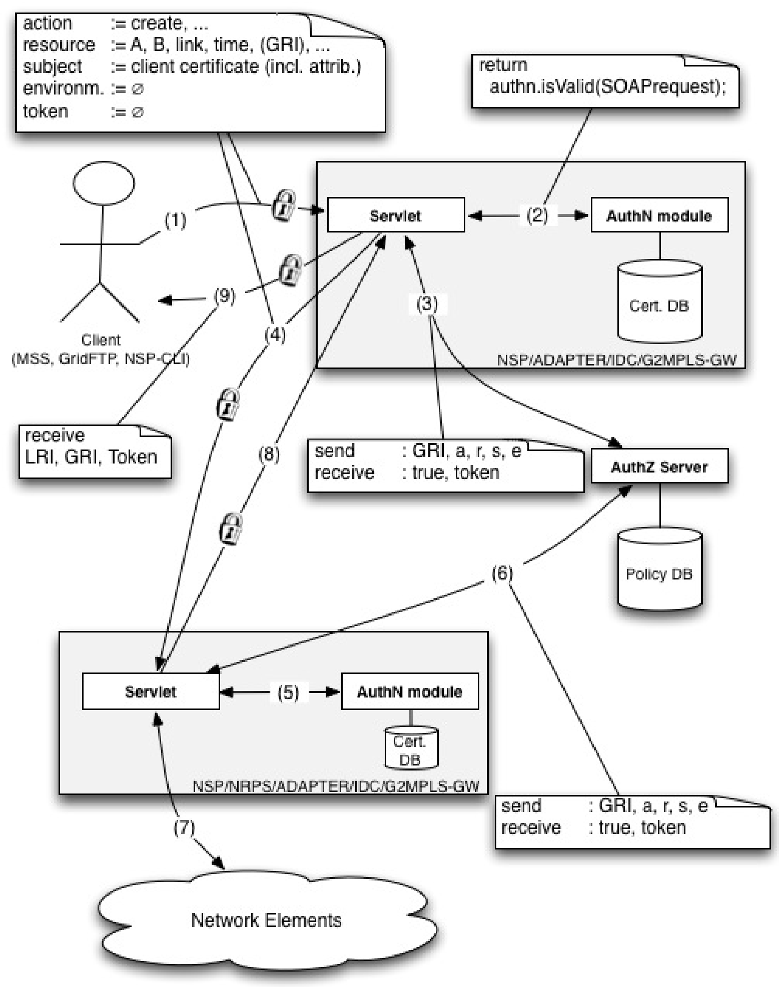


Figure 4.7: Generalized AuthN/AuthZ workflow for path creation

4.4.3.2 *Path administration*

In **Figure 4.8** a generalized AuthN/AuthZ workflow for a path administration process is depicted. The different steps can shortly be described as follows: (1) The client – in this case for example the MSS – wants to activate [action| a reservation identified by the GRI in the [token]. The request is signed by the client's certificate [credentials] and encrypted with the NSPs/IDCs/G^{2}MPLS-GWs public key. (2) The NSP/IDC/G^{2}MPLS-GW validates the signature of the incoming message by comparing it with the pre-installed public keys. (3) After the successful authentication, parts of the request will be sent to a Token Validation Service (TVS) by using the GAAA-tk. The information sent to the TVS consists of the token and the user's credentials. The TVS in return will send a boolean value. (4) Now the NSP/IDC/G^{2}MPLS-GW creates a new activation request for the involved systems including the token. (5)(6) The next system now runs the AuthN and AuthZ process again for the incoming request. (7) Thereafter the underlying network elements (NEs) are configured and a confirmation is sent back to the NSP/IDC/G^{2}MPLS-GW in step (8). (9) Finally the client receives the confirmation for the activation.

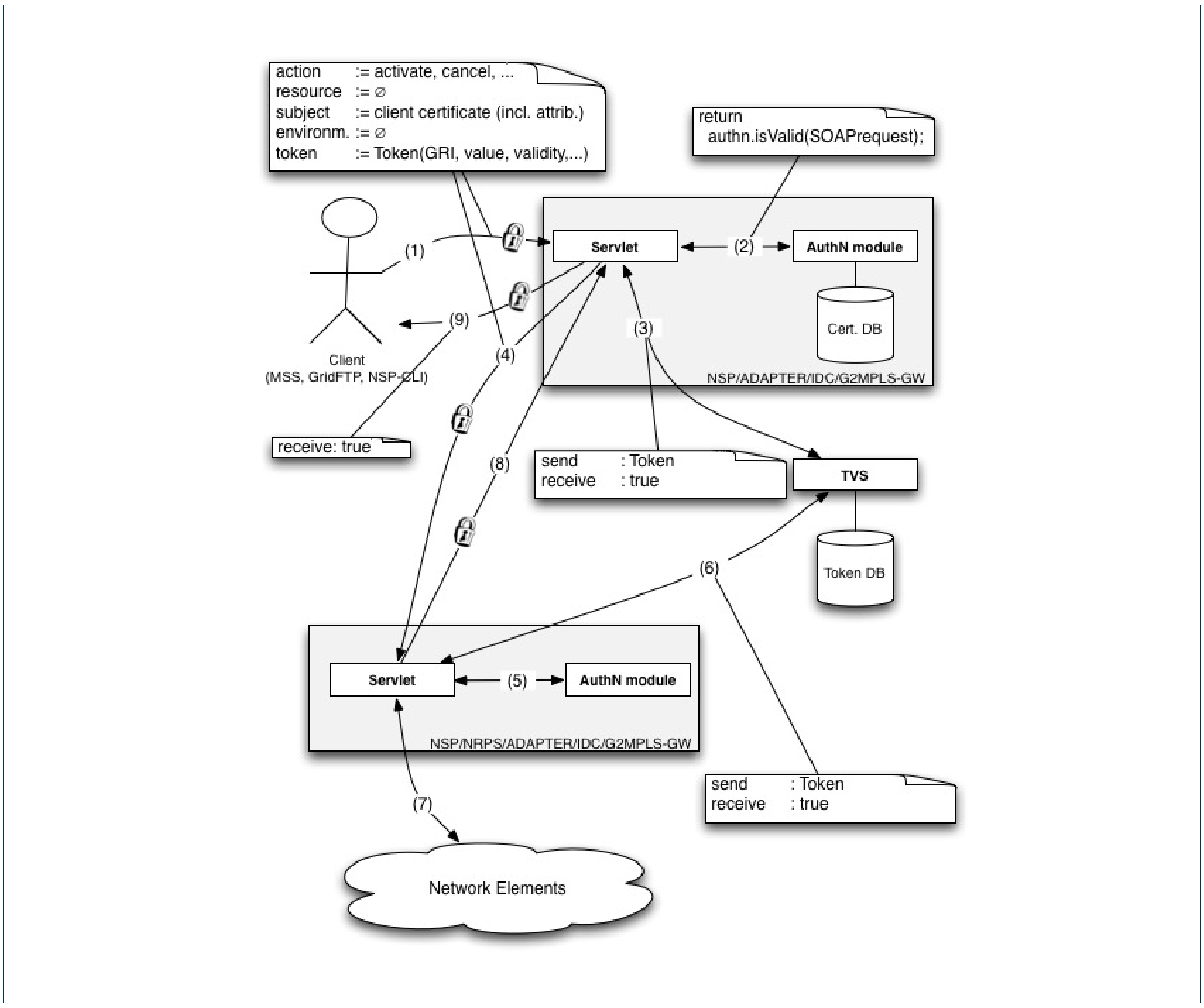


Figure 4.8: Generalized AuthN/AuthZ workflow for path administration

5 Conclusions

5.1 Summary

This deliverable documents a series of integration tests of the Network Service Plane implementation which links different NRPS controlled domains (via NRPS Adapters) and a GMPLS domain (via a “Thin NRPS”) with a Metascheduler or any other requestor to whom the multidomain nature of the underlying network is hidden. Prior to the description of the actual tests, the scenario (i.e. the WP1 testbed) has been described.

The tests have proven that in the prototypical implementation, the NSP is able to compose interdomain advance reservations of several intradomain advance reservations requested from the NRPS Adapters in each of the participating domains. The interdomain path is dynamically chosen depending on the availability of resources within the domains which is also queried through the NRPS Adapters.

Finally, basic approaches for an integration of NSP and G^2MPLS domains, their advantages, disadvantages, and problematic issues have been identified and discussed. From our understanding, this integration is possible, and implementation thereof should begin when first a G^2MPLS is operational and running in the testbed.

5.2 Future work

Though the NSP implementation has been proven to work in the tests described here and during several demonstration activities, there are several enhancements that are planned to be implemented in the following months.

One goal is to support an advanced hierarchical architecture. In such an architecture, a central NSP instance controls not only a number of NRPS Adapters, but also NSP subdomains that aggregate several subdomains themselves. In a purely hierarchical model, there still needs to be a “root NSP” at the highest hierarchy level. To allow for a peering model with other systems (i.e. G^2MPLS), a peering model will therefore also be implemented for communication within the “root NSP”, allowing several domains to peer at the highest hierarchy level.

The current NSP implementation only supports fixed (immediate and advance) reservations, i.e. reservations with fixed starting and ending times. This reservation type is suitable for applications such as collaborative working or distributed simulations, where the availability of Grid resources is known in advance and the amount of bandwidth is fixed. In practise, another reservation type is relevant, namely the “malleable reservations”. The constraints defined by this reservation type are a specific amount of data and a specific time window within which this amount of data has to be transferred. These data transfers are necessary e.g. for distributing input data for a distributed simulation and for collecting the results afterwards.

An efficient communication in this context is needed in order to reduce the response time for the numerous messages. Currently, the NSP (and in case of a hierarchical model the different NSPs) and NRPS Adapters are polling status information from the underlying systems. A notification framework will be implemented to push status changes to higher layer instances, thereby lowering delays and overhead introduced through polling.

In addition, a more matured contraint management system will be implemented. The NSP will be enhanced to deal with interdomain links of various contraints (e.g. bandwidth and delay). Another development will be the adaptation of the NSP to handle heterogenious technologies for interdomain paths. The path finding will take into account multiple constraints and technologies that are available.

Besides this, the approach for token based security mechanisms on the network layer chosen by WP4 will be supported by the NSP and the NRPS Adapters. The focus will be on transitively forwarding token/security related authentication and authorisation information from the WP3 Middleware down to each NRPS in case of a path creation. The authorisation of path configuration requests (e.g. activation or termination) will support the WP4 token model as well.

To allow the NSP to communicate with different BoD systems (besides G^2MPLS), an interoperable interface needs to be developed. Therefore, other research projects which may bring valuable input to Phosphorus will be contacted. This interoperation may include as well several changes in the NSP to adapt it to the messaging architecture of the other systems.

Furthermore, some practical issues need to be solved. First, the error resiliency of the NSP implementation should be increased. Currently, a single NRPS Adapter that is not working properly can cause instability of the complete system under specific circumstances. Second, the used Web service stack and data-binding libraries seem to delay the message flow between the participating systems in an inappropriate manner. Finally, some long-term and stress tests are needed to identify and remove memory leaks and scalability issues.

6 References

[JUnit]	http://www.junit.org/
[Phosphorus-D1.4]	Phosphorus Deliverable 1.4: “Definition and Development of the Network Service Plane and Northbound Interfaces Development”.
[Phosphorus-D6.1]	Phosphorus Deliverable 6.1: “Testbed Design”.
[RFC1918]	Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, E. Lear: “Address Allocation for Private Internets”. IETF RFC 1918, February 1996.
[Phosphorus-D2.7]	Phosphorus Deliverable 2.7: “Grid-GMPLS network interfaces specification”
[tinc]	http://www.tinc-vpn.org/
[rampart]	http://ws.apache.org/rampart/