034115

PHOSPHORUS

Lambda User Controlled Infrastructure for European Research

Integrated Project

Strategic objective:
Research Networking Testbeds

Deliverable D6.8-update
New developments testing report

Due date of deliverable: 2009-06-30
Actual submission date: 2009-06-30
Document code: Phosphorus-WP6-D6.8-update

Start date of project:
October 1, 2006

Duration:
33 Months

Organisation name of lead contractor for this deliverable:
Instytut Chemii Bioorganicznej PAN

Release 2

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 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)	

Authors

Artur Binczewski	PSNC
Wojbor Bogacki	PSNC
Gino Carrozzo	NXW
Nicola Ciuli	NXW
Mihai Cristea	UvA
Yuri Demchenko	UvA
Thomas Eickermann	FZJ
Eduard Escalona	UEssex
Jordi Ferrer Riera	I2CAT
Sergi Figuerola	i2CAT
Joan Antoni Garcia Espin	I2CAT
Marcin Garstka	PSNC
Damian Parniewicz	PSNC
Maciej Stroiński	PSNC
Jan Węglarz	PSNC
Alexander Willner	UoB
Wolfgang Ziegler	FhG

Abstract

PHOSPHORUS addresses some of the key technical challenges to enable on-demand e2e network services across multiple domains. The concept of the PHOSPHORUS network will make applications aware of their complete GRID resources (computational and networking) environment and capabilities, and able to make dynamic, adaptive and optimized use of heterogeneous network infrastructures connecting various high-end resources. PHOSPHORUS will enhance and demonstrate solutions that facilitate vertical and horizontal communication among applications middleware, existing Network Resource Provisioning Systems, and the proposed GRID-GMPLS Control Plane.

One of the main assumptions of PHOSPHORUS is that the project propositions and developments should be validated and demonstrated in a real advanced optical network. To achieve this, the project has built a distributed testbed in which the project outcome is demonstrated with a set of real scientific applications in a set of real-life scenarios.

This document summarises the results of testing and validation of the project developments which were done during the whole course of the project. It is an update to a deliverable D6.8 submitted in September 2008 which described tests done before this date.

Table of Contents

List of Figures

List of Tables

Executive Summary

PHOSPHORUS addresses some of the key technical challenges to enable on-demand e2e network services across multiple domains. The PHOSPHORUS network will make applications aware of their complete GRID resources (computational and networking) environment and capabilities, and able to make dynamic, adaptive and optimized use of heterogeneous network infrastructures connecting various high-end resources. PHOSPHORUS will enhance and demonstrate solutions that facilitate vertical and horizontal communication among applications middleware, existing Network Resource Provisioning Systems, and the proposed GRID-GMPLS Control Plane.

One of the main assumptions of PHOSPHORUS is that the project propositions and developments should be validated and demonstrated in a real advanced optical network. To achieve this, the project has built a distributed testbed in which the project outcome is verified with a set of real scientific applications in a set of real-life scenarios. This way the testbed emulates a modern GRID environment in which demanding applications running on computational nodes use a transmission network to exchange data between the nodes and access external devices.

This document summarises the results of testing and validation of the project developments which were done during the whole project duration. It is an update to a deliverable submitted in September 2008 which described tests done before this date.

The PHOSPHORUS developments which were tested in the testbed come from different PHOSPHORUS activities. Tests referring to developments of each workpackage are described in separate chapters. Chapter one describes tests of the HARMONY software developed by Workpackage 1 which integrates the different Network Resource Provisioning Systems existing before the start of PHOSPHORUS (UCLP, DRAC, ARGON). Chapter two describes the tests of the GRID-GMPLS Control Plane (G^{2}MPLS) prepared by Workpackage 2 which extends the GMPLS services towards GRIDs. The tests were conducted in an optical domain as well as in an Ethernet domain containing some Layer 2 switches. The next chapter describes tests of applications and GRID middleware (developed by Workpackage 3), while chapter four focuses on a test of the security mechanisms (GAA-TK) developed by Workpackage 4.

The subsequent chapters describe tests in which the integration of developments of different workpackages was validated. Each chapter describes validation of integration of developments of a pair of workpackages. Those tests were conducted to verify that the developments of different PHOSPHORUS activities can be integrated together and will make an integrated system which will benefit the research community. Chapters 6, 7 and 10 are of particular interest as they describe the tests in which real scientific applications were used to validate developments of the project.

All the tests described in this document were conducted in a distributed PHOSPHORUS testbed with the use of the resources available in different local testbeds. The testbed is distributed across Europe and North America to realistically represent an optical network of a global scope. The local testbeds in which the switching and GRID resources are placed are connected through several different infrastructures (GÉANT2, CAnet, European NRENs, cross-border fibres) to make a single distributed testbed. Each test was conducted in one or more local testbeds – according to the needs for infrastructure which were determined by the test scenario.

Some of the tests have been previously described in deliverables prepared by the work packages which submitted their developments for tests. They are described also in this deliverable in order to make the search for information about tests easier – the information can be found in this deliverable as well as in deliverables describing the developments of the other workpackages.

The results of the tests were used by the workpackages which provided their developments for tests in elimination of bugs, planning next phases of developments as well as proving the ideas behind the developments in a real network.

1 NRPS experiments – WP1

1.1 Tested features of the Harmony system

Since its first description in D1.4 [PHOSPHORUS-D1.4], the Network Service Plane implementation has been extended by several features that required testing, including the integration with the Meta-Scheduling System (MSS) and the G^2MPLS developments and the extensions described in D1.8 [PHOSPHORUS-D1.8]. In this chapter, the tests of the peer-to-peer topology distribution, peer-to-peer reservation setup and malleable reservations and their corresponding results are described. The following subsections provide a short introduction to the tested features.

1.1.1 Peer-to-peer topology distribution

In the new peer-to-peer operation mode, there is not central instance where all domains register their domain information. Instead, the domain information is flooded to all domains. Each domain registers its information with its peers in regular time intervals. Upon reception of new information from other domains, this information is also forwarded.

The decision whether received domain information is newer than the information in the database is based on a sequence number field that is incremented by the originator of the domain information and on the time when the last domain information was received, the latter in order to recover from the case where an instance is e.g. rebooted and did not store its last sequence number correctly. So the update information is rejected if the sequence number is smaller than the current one and if the difference between current registration time and the last registration time is less than 15 minutes.

1.1.2 Malleable reservations

While fixed reservations with the parameters “starting time” and “duration” are tailored for “interactive” applications, another important application type requires the possibility to make data transfers within a given deadline. For this reservation type, the parameters “earliest starting time”, “deadline”, “data amount” (and “minimum” / “maximum bandwidth”) specify a time window within which the data transfer has to take place. It offers more degrees of freedom, reflecting that it is irrelevant for the application when exactly the transfer takes place and what specific bandwidth is used. This takes the burden of trying to find a time window when the necessary resources are available from the application and places it to the service plane.

1.2 Testbed setup

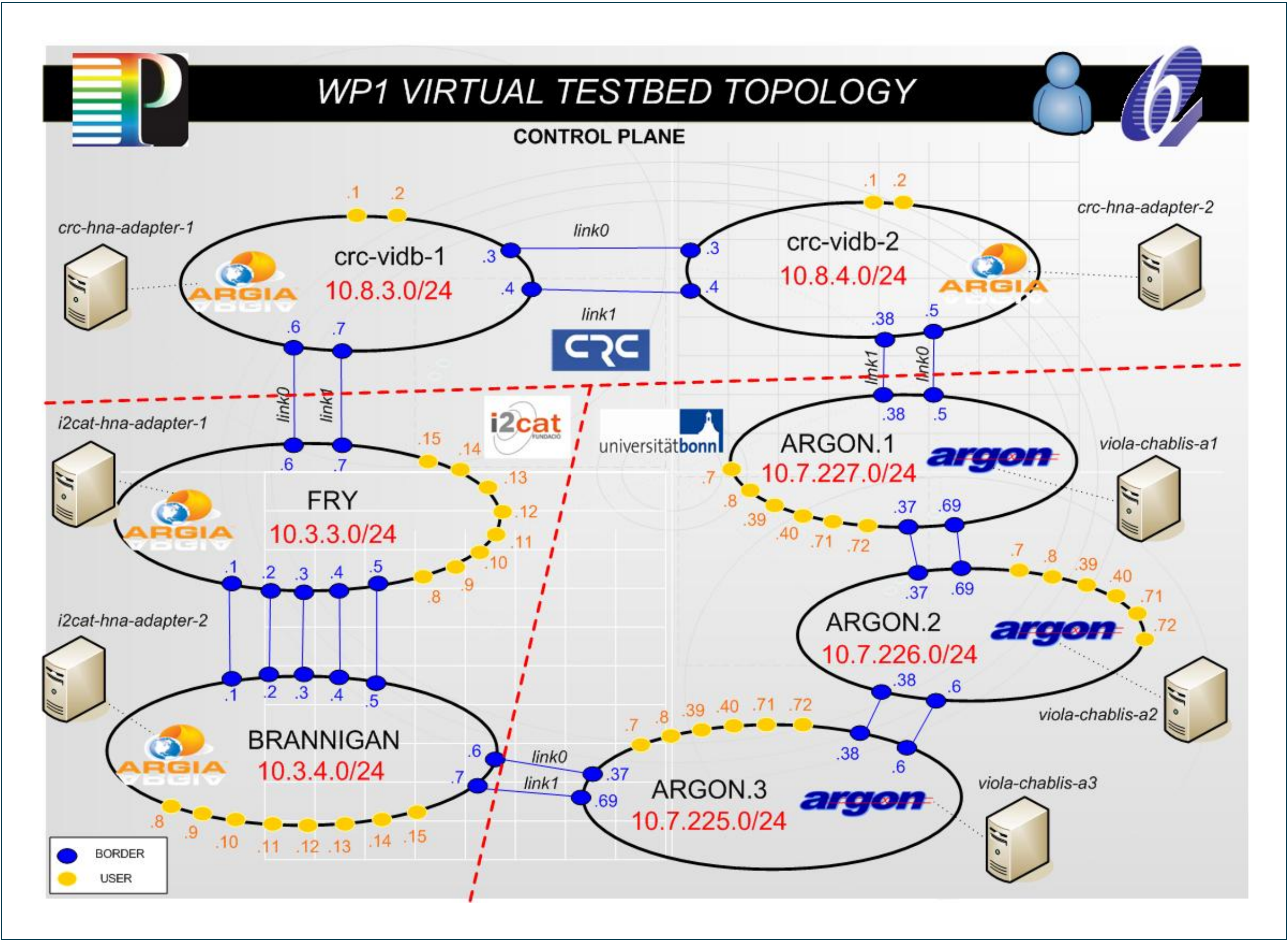

Figure 1.1: WP1 related layers

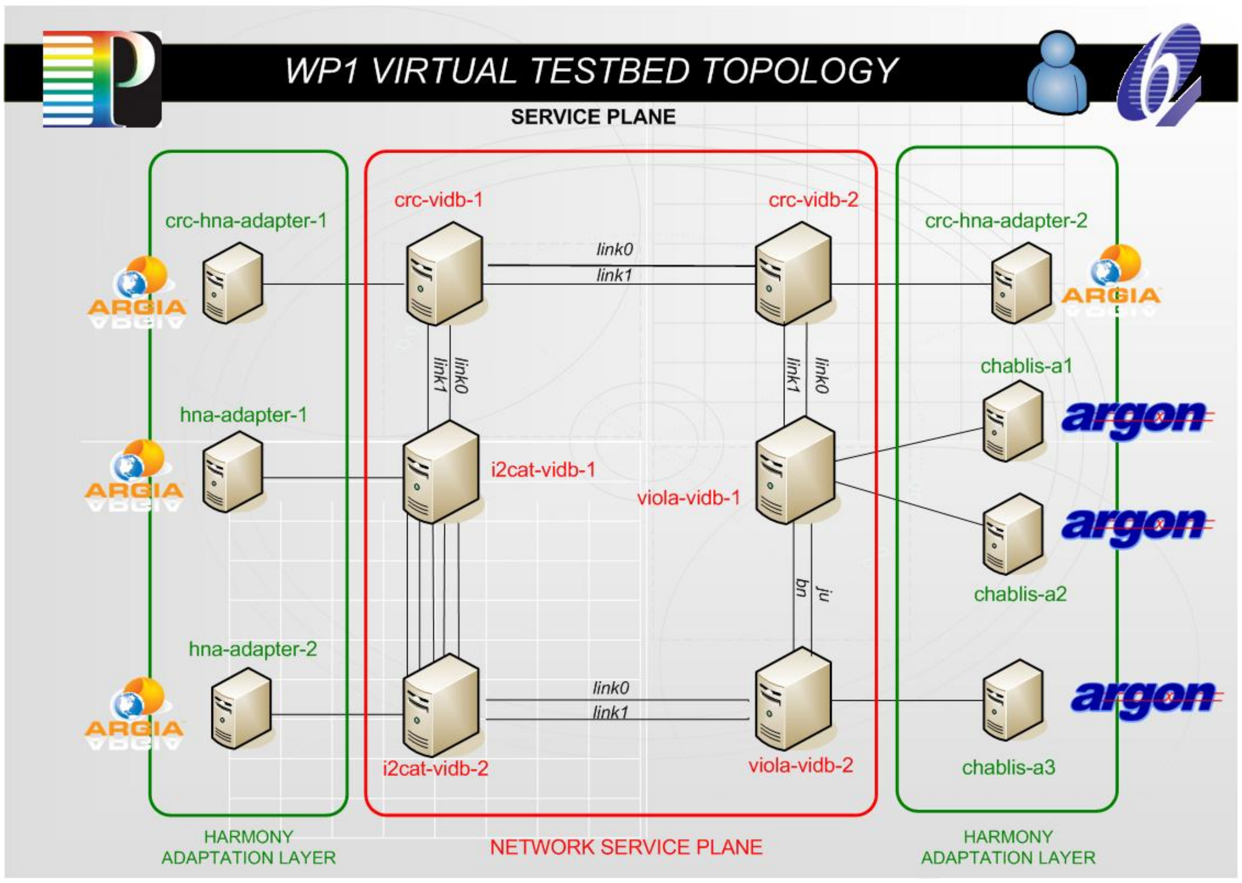

Figure 1.2: WP1 related layers

Figure 1.1 and Figure 1.2 show the virtual testbed setup in which the tests have been carried out. It has been decided not to use the existing testbed with physical links because the tests mainly concern the behaviour of the service plane, so the data plane is not relevant for the tests. From the service plane point of view, there is no difference between the virtual testbed and the real testbed, because communication takes place with the same NRPSs. These NRPSs are merely operated in a test mode in which they do not control real devices, but "dummy" devices.

Each of the partners participating in the virtual testbed (i2Cat, University of Bonn, and CRC) operate two Inter-Domain Brokers (IDBs), each of which is connected to two other IDBs, forming a ring topology. This topology is motivated by the need to have some longer routes, also for testing topology information dissemination. Each IDB controls a single Harmony NPRS Adapter (HNA), except for one of the IDBs at the University of Bonn that controls two ARGON sub-domains. The general architecture is described in detail in Deliverable 1.4 and Deliverable 1.8.

Descriptor	Test connection	Blocking connections
i2cat	10.3.3.8 – 10.3.4.8	10.3.3.11 – 10.3.4.11 10.3.3.12 – 10.3.4.12 10.3.3.13 – 10.3.4.13 10.3.3.14 – 10.3.4.14 10.3.3.15 – 10.3.4.15
viola	10.7.225.7 – 10.7.226.7	10.7.225.71 – 10.7.226.71 10.7.225.72 – 10.7.226.72
crc	10.8.3.1 – 10.8.4.1	10.8.3.2 – 10.7.227.72 10.3.3.15 – 10.8.4.2
i2cat-viola	10.3.4.8 – 10.7.226.7	10.3.4.14 – 10.7.225.71 10.3.4.15 – 10.7.225.72
i2cat-crc	10.3.4.8 – 10.8.3.1	10.3.3.14 – 10.8.4.1 10.3.3.15 – 10.8.3.2
viola-crc (1)	10.7.226.7 – 10.8.3.1	10.7.226.71 – 10.7.227.71 10.7.226.72 – 10.7.227.72
viola-crc (2)	10.7.226.7 – 10.8.3.1	10.7.227.71 – 10.8.4.1 10.7.227.72 – 10.8.4.2

Table 1.1: Test connections and corresponding blocking connections

Table 1.1 shows the test connections that have been used in the following tests, together with additional "blocking connections" that have been setup prior to the actual test connection in some test to force the test connection to be established across the longer path in the cycle of domains.

1.3 Test results

1.3.1 Topology information dissemination test

To test the flooding of domain information in the distributed architecture, information of a new (dummy) domain has been injected at one domain (viola-vidb-2). Table 1.2 shows the time offsets at which this new domain information has been added to the other domains' databases. The random time offsets are due to the fact that the advertisement of current topology information to peer domains is triggered 5 minutes (tradeoff between overhead and robustness) after the last advertisement has been completed.

Time offset [s]	Domain
0	viola-vidb-2
109	i2cat-vidb-2
140	viola-vidb-1
398	i2cat-vidb-1
438	crc-vidb-2
677	crc-vidb-1

Table 1.2: Detailed results of the fixed reservation tests (single measures)

Each domain has added this new domain information to its database and forwarded it to each of its peers exactly once. Duplicates have been detected and were discarded. Therefore, this test has shown that the topology dissemination mechanism is functional.

1.3.2 Fixed reservation tests

For the fixed reservation tests, an immediately starting reservation between two endpoints was made. After a delay of 20s, it was verified that the status of the reservation is ACTIVE. Then, an availability request for the same two endpoints and the same starting time was made and it was verified that the reply was ENDPOINTS_NOT_AVAILABLE.

Connection descriptor	Blocked endpoints	Alternative start time offset
i2cat	-	360s
viola	10.7.225.7 10.7.226.7	325s
crc	-	0s
i2cat-viola	10.7.226.7	360s
i2cat-crc	-	360s
viola-crc	10.7.226.7	329s

Table 1.3: Detailed results of the fixed reservation tests

These tests were basically successful, but have shown some minor deficiencies. For one, not all systems actually return a list of blocked endpoints, even if the result code is ENDPOINTS_NOT_AVAILABLE. Second, the CRC domain failed to report an alternative start time offset. However, it should be noted that this is not a required, but an optional feature.

1.3.3 Malleable reservation tests

For testing malleable reservations, two different test types were defined. Both create fixed reservations that block the shortest routes prior to requesting a malleable reservation. In the first case, the duration of the blocking reservations is short enough such that the malleable reservation can be made along the shortest route after the fixed reservations have expired. In the second case, the fixed reservations are active until the deadline of the requested malleable reservation, so that the malleable reservation has to take a longer route.

1.3.3.1 *Deferred malleable reservation tests*

For these tests, the blocking reservations were fixed reservations with 60 minutes duration that were started immediately. The earliest start time for the malleable reservations was set to 30 minutes after the start time of the blocking reservations, and the deadline was set to 90 minutes after the earliest start time.

Consequently, a time window of 60 minutes was left for the malleable reservations, and the earliest possible actual starting time of the malleable reservation is 30 minutes after the start time specified in the request. Since the requested data amount to be transferred was 100GB (100 GigaBytes) and the requested maximum bit rate was 1Gbps (1 Gigabit per second), the actual duration necessary for the transfer was 800s, which fits into this time window well.

Connection descriptor	Involved domains	Start delay
i2cat		
viola	viola-chablis-a3 viola-vidb-1	30 minutes
crc	crc-vidb-1 crc-vidb-2	60 minutes
i2cat-viola	i2cat-vidb-2 viola-chablis-a3 viola-vidb-1	60 minutes
i2cat-crc	crc-vidb-1 i2cat-vidb-1 i2cat-vidb-2	60 minutes
viola-crc (1)	crc-vidb-1 crc-vidb-2 viola-vidb-1	0
viola-crc (2)	crc-vidb-1 crc-vidb-2 viola-vidb-1	60 minutes

Table 1.4: Detailed results of the deferred malleable reservation tests

Table 1.4 shows the domains involved in the malleable reservations for each of the test connections and the start delay with respect to the earliest start time specified in the malleable reservation request. It can be seen that this delay is 60 minutes in most cases, due to the fact that the algorithm for creating malleable reservations in the IDB code assumes an alternative start time offset of 60 minutes if a service is reported to be unavailable, but no appropriate alternative start time offset is specified in the reply. Obviously, only the systems in the domains at the University of Bonn respond with appropriate alternative start time offsets.

The reason for the start delay of 0 for the “viola-crc (1)” connection is an important conclusion from the tests that have been performed. On first sight, it seems that this result is not possible, because the only routes between the two endpoints of the test connection that traverse the domains listed in the table are blocked by the blocking connections for a start delay of 0. A closer look at the connections reveals the following: The IDB receiving the original request computes the following inter-domain path:

10.7.226.7 – 10.7.227.5, 10.8.4.5 – 10.8.4.3, 10.8.3.3 – 10.8.3.1

Then, viola-vidb-1 checks the availability of 10.7.226.7 – 10.7.227.5. Since the two inter-domain links between its ARGON sub-domains are blocked, it finds an alternative path:

10.7.226.7 – 10.7.226.38, 10.7.225.38 – 10.7.225.37, 10.3.3.4 – 10.3.3.6, 10.3.4.6 – 10.3.4.4, 10.8.3.6 – 10.8.3.4, 10.8.4.4 – 10.8.4.38, 10.7.227.38 – 10.7.227.5

Therefore, the complete path that is established consists of 9 connections on NRPS level.

This is clearly an undesirable behaviour that is caused by the fact that the search for a feasible path should involve peer domains only at the highest hierarchy level and should then be restricted to sub-domains once this level is left. WP1 is currently planning a solution for this problem.

Aside from this, the tests show that the malleable reservation handler is able to find an available time slot for a reservation, if resources are partly blocked by other reservations.

1.3.3.2 *Rerouted malleable reservation tests*

The test setup of the following tests is the same as for the tests described in the previous section, except for the duration of the blocking reservations which was extended to 120 minutes. Therefore, the malleable reservation could not be made on the shortest path.

Connection descriptor	Involved domains	Start delay
i2cat-viola	crc-vidb-1 crc-vidb-2 i2cat-vidb-1 i2cat-vidb-2 viola-vidb-1	0
i2cat-crc	crc-vidb-1 crc-vidb-2 i2cat-vidb-2 viola-chablis-a3 viola-vidb-1	0
viola-crc (1)	crc-vidb-1 crc-vidb-2 viola-vidb-1	0
viola-crc (2)	crc-vidb-1 i2cat-vidb-1 i2cat-vidb-2 viola-chablis-a3 viola-vidb-1	0

Table 1.5: Detailed results of the rerouted malleable reservation tests

Table 1.5 shows the results of these tests. For the same reason described in the previous section, the connection for “viola-crc (1)” seems to take a shortest path, which is not true because the requested connection between the endpoints in viola-vidb-1 is a multi-domain connection traversing all of its peer domains.

These tests show that the malleable reservation is handler is able to use alternative routes when resources are blocked by other reservations.

1.4 Performance and Scalability tests of the Harmony NSP

Several performance and scalability tests have been realized aiming to validate Harmony capabilities by analyzing the time responses on performing multi-domain request across the different NRPSs and analyzing the stress (in terms of numbers of simultaneous requests) supported by the different service plane architectures.

1.4.1 Formal definitions

This section describes some formal definitions which may help the understanding of the tests and their results

1.4.1.1 *Request processing durations*

The service plane is described as the digraph $G_s = \ H,C$ with *H* a set whose elements are called *Harmony Service Interface entity* (HSI entity), and *C* a set of ordered pairs of vertices, called *HSI interconnection* with $u_s, v_s \ \in C$. In the centralized or hierarchical model these HSI interconnections also reflect the hierarchy. A path in G_s is denoted as $p_s = \ u_{s_0}, v_{s_0} \ ,..., \ u_{s_{m-1}}, v_{s_{m-1}}$ with $m \in$ N and $len \ p_s \ = m$. The child of a HSI adapter is defined as $s_{u_s} = \ w_s \mid \ u_s, w_s \ \in C$. This means, that the HSI entity u_s delegates the reservation task to the set $s \ u_s$. Thus we have a tree structure of HSI entities. In the distributed model we have a mesh of HSI entities which may be themselves the root of a HSI entity tree.

Let the set of Domains for which an HSI entity *h* is responsible be $r \ h \in H \ \in D$ and $R \ H_i \subseteq H \ = \ r \ h \ | h \in H_i$. The set of responsible HSI entities of a data path *d* is $R \ a \ p_d$. The time a single HSI entity $h \in R \ a \ p_d$ needs to process a request internally is denoted as t_h and delegates the reservation to its child HSI entities. If we assume that all children of *h* are requested consecutively, we can define t_h^{total} of a HSI entity recursively as:

(4.1)
$$t_h^{total} = \begin{cases} t_h & if\ |s(h)| = 0 \\ t_h + \sum_{h_i \in s(h) \cap R(a(p_d))} t_{h_i}^{total} & if\ |s(h)| \geq 1 \end{cases}$$

If the communication with the children is parallelized, we can define $t_{h_{total}}$ as:

(4.2)
$$t_h^{total} \begin{cases} t_h & if\ |s(h)| = 0 \\ t_h + \max_{h_i \in s \ h \ \cap R \ a \ p_d} t_{h_i}^{total} & if\ |s(h)| \geq 1 \end{cases}$$

If a distributed model for the service plane is used there is a mesh of HSI entities on top level $H_{top} \in H$. Each top level HSI entity can itself be the root of a hierarchical HSI entity structure. Thus, we get for path request for a distributed service plane operating in a consecutive mode:

(4.3)
$$t^{total} = \sum_{h_i \in H_{top} \cap R \ a \ p_d} t_{h_i}^{total}$$

For a path request for a distributed service plane operating in a parallel mode:

(4.4)
$$t^{total} = \max_{h_i \in H_{top} \cap R \ a \ p_d} t_{h_i}^{total}$$

1.4.1.2 *Preliminary probability studies*

This section describes the different probabilities of forwarding or not forwarding a request when the Network Service Plane is composed of several Inter-domain Brokers (IDBs) working in hierarchical approach or distributed approach. If we assume the hierarchical scenario depicted in figure 1.4 when a requests gets and IDB there are two possibilities:

1. The resources are under control of the IDB

(4.5) $$P(Success) = p_s$$

2. The resources are not under the control of the IDB

(4.6) $$P(Failure) = p_f = 1 - p_s$$

In case 1, the resources can be totally or partially under control of the IDB; that is, the IDB which receives the request can control both source and target TNA or can control only one. In case that both

(4.7) $$P(S\_Both) = p_{sb}$$

(4.8) $$P(S\_Part) = p_{sp}$$

(4.9) $$p_s = p_{sp} + p_{sb}$$

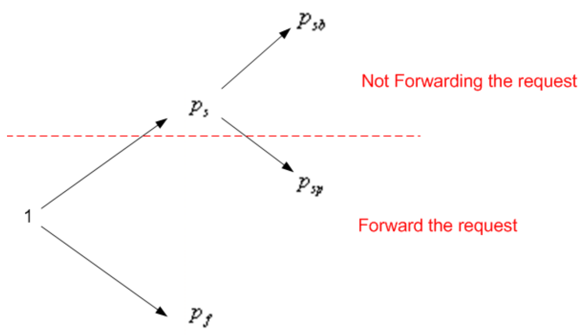

Figure 1.3: Probabilities tree

Consider that there are *n* transport network addresses and that each domain at the **same level of the hierarchy** controls the same number of endpoints. Consider also the worst case: that is, a request arrives to an IDB of the lower level of the hierarchy and the resources requested are located in the opposite side of the three, as depicted in Figure 1.4, where the circles represent IDBs, the square HNA and the yellow circles transport network domains.The flow of the requests is depicted in next figures.

(4.10)
$$P\left(TNA \in IDB_L\right) = \frac{1}{n}$$
$$P\left(TNA \in IDB_{L-1}\right) = \sum_i P_i\left(TNA \in IDB_L\right)$$
$$\ldots$$
$$P\left(TNA \in IDB_1\right) = 1$$

Expression 4.11 shows how the probability to successfully find a TNA in one IDB increases as the IDB is located in upper levels of the hierarchy (i.e. the IDB which is in the first level has probability 1, since it has knowledge of the whole topology).

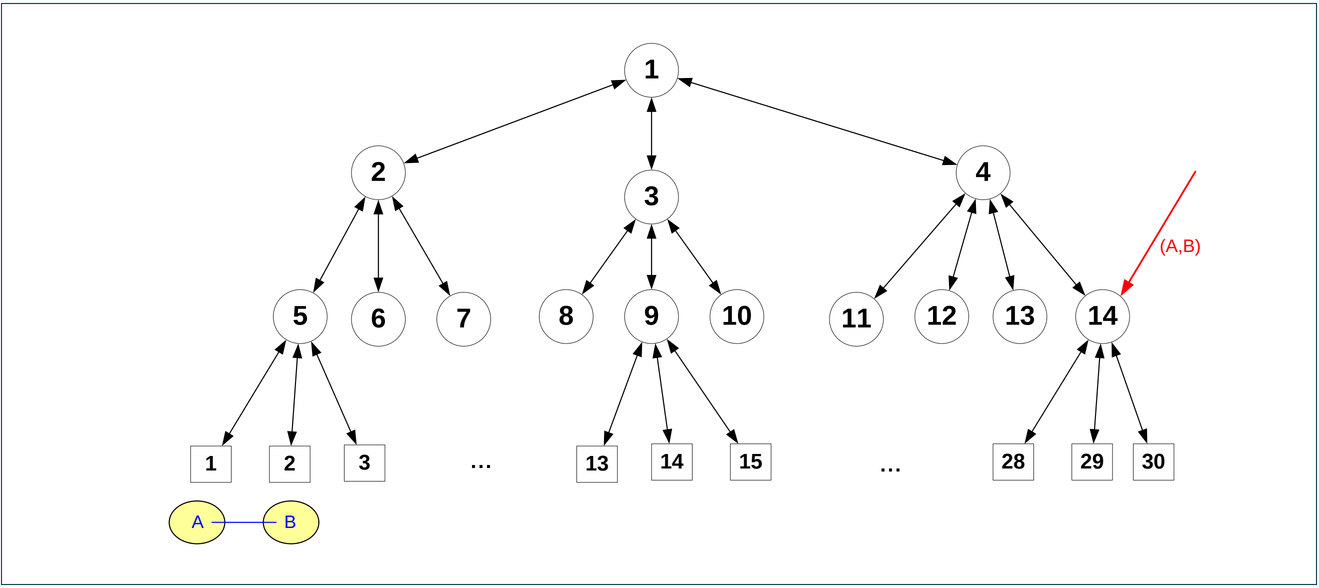

Figure 1.4: A request is received in IDB 14. Source TNA of the request is A and target TNA is B.

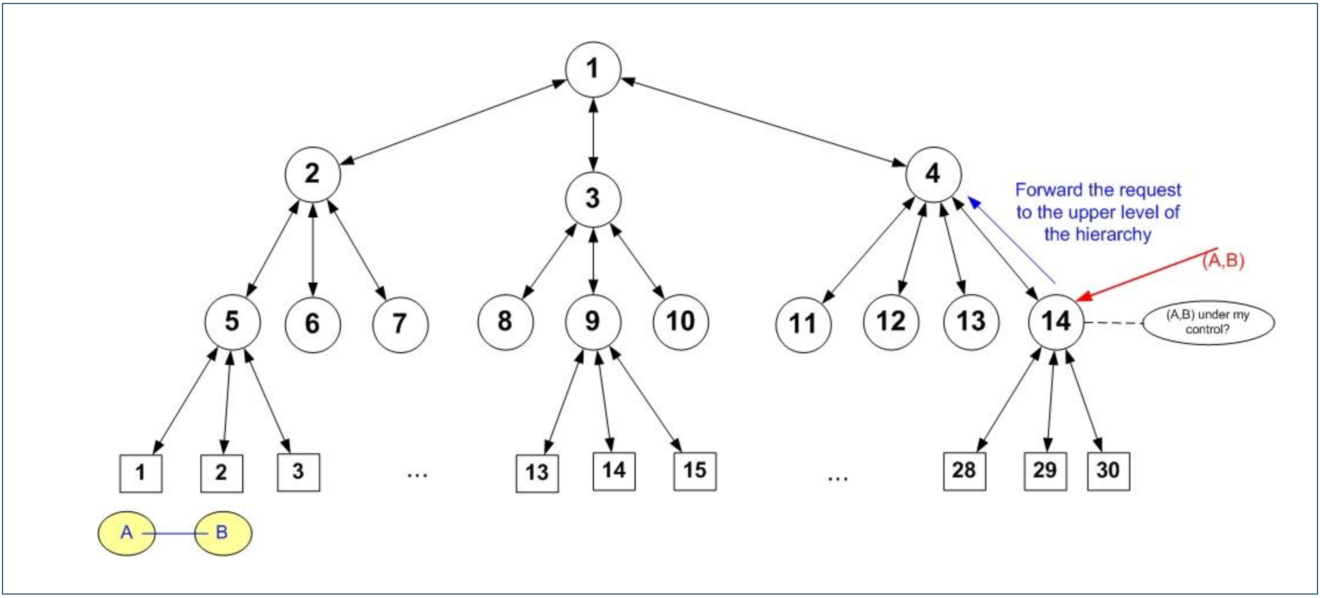

Figure 1.5: IDB 14 does not control the resources requested. Consequently, it forwards the request to the upper level.

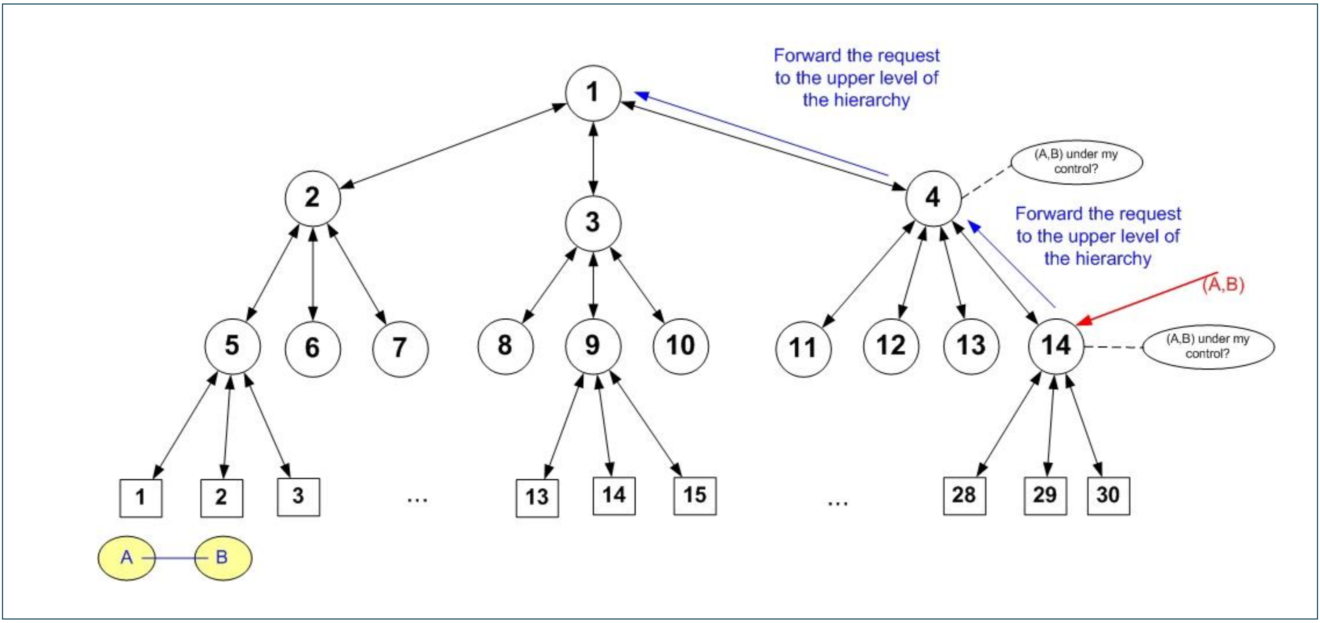
Forward the request to the upper level of the hierarchy
1
2
3
4
(A,B) under my control?
Forward the request to the upper level of the hierarchy
(A,B)
5
6
7
8
9
10
11
12
13
14
(A,B) under my control?
1
2
3
...
13
14
15
...
28
29
30
A
B

Figure 1.6: IDB 4 does not control the requested resources. Consequently, it forwards again the request to the upper level

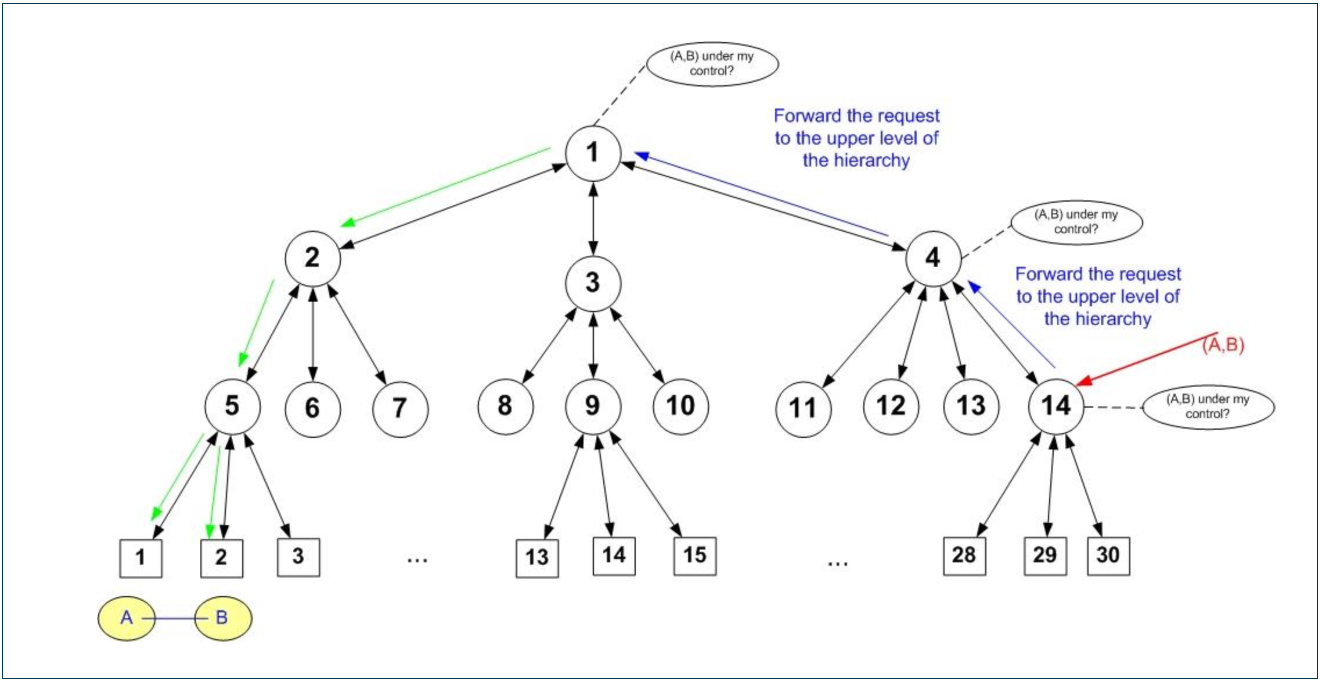

Figure 1.7: The request is processed from IDB 1, since it has whole topology knowledge.

Considering this case and the previous considerations, let's study the distinct probabilities:

(4.11) $$P\ \ A \in IDB\ \ = \frac{1}{n} = P(B \in IDB)$$

(4.12) $$P\ \ A \in IDB \cap B \in IDB\ \ = P\ \ A \in IDB\ \times P\ \ B \in IDB / A \in IDB$$

(4.13) $$P\ \ \ A \cap B\ \ \in IDB\ \ = \frac{1}{n} \times P\ \ B \in IDB / A \in IDB$$

(4.14) $$P(A \in IDB \cup B \in IDB) = P(A \in IDB) + P(B \in IDB) - P(A \in IDB \cap B \in IDB)$$

(4.15) $$P\ \ A \notin IDB\ = P\ \ B \notin IDB\ \ = \frac{n-1}{n}$$

(4.16) $$P\ \ \ A \cap B\ \ \notin IDB\ = P\ \ A \notin IDB\ \times P\ \ B \notin IDB / A \notin IDB\ \ = \frac{n-1}{n} \times P\ \ B \notin IDB / A \notin IDB$$

(4.17) $$p_f = P(A \notin IDB \cap B \notin IDB)$$

(4.18) $$p_{sb} = P\ \ A \in IDB \cap B \in IDB\ \ = \frac{1}{n} \times P\ \ B \in IDB / A \in IDB$$

To sum up, the probability of having to forward the request is determined by p_{sp} and p_f, while the probability of processing the request in the IDB is determined by p_{sb}. Please note that this preliminary study only stands for valid requests. In this sense, if one request contains one TNA not contained in the Network Service Plane the study is not useful.

Regarding the distributed operating mode within the NSP, the probabilities study remains the same, but the distinct behaviour comes out when forwarding a request. In this sense, when the service plane is working in distributed operating mode, the flooding algorithm guarantees that each IDB has the knowledge of the whole service plane topology in its database. Thus, there will be only one-level-request forwarding.

1.4.2 Performance tests

This section shows the results of the different performance tests executed over the Network Service Plane.

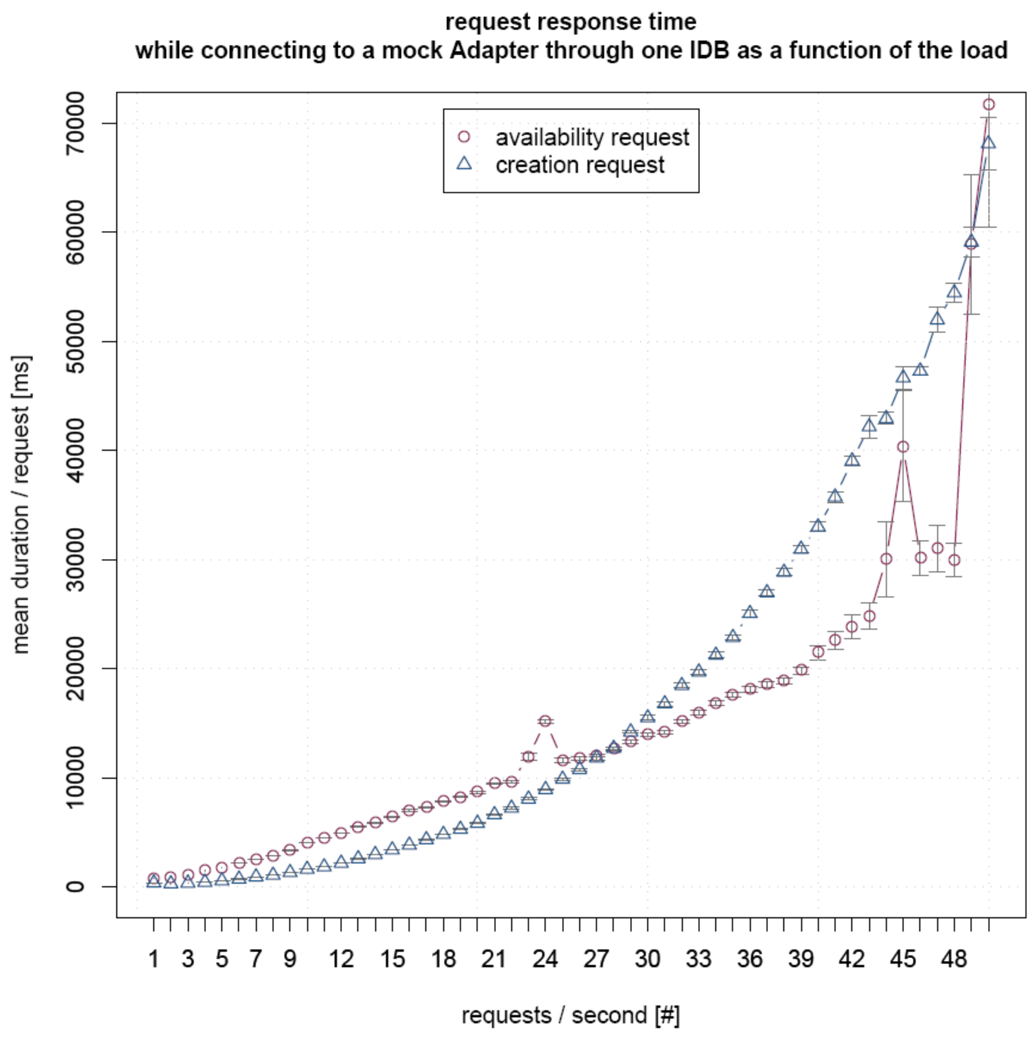

Figure 1.8: Request response time with one IDB

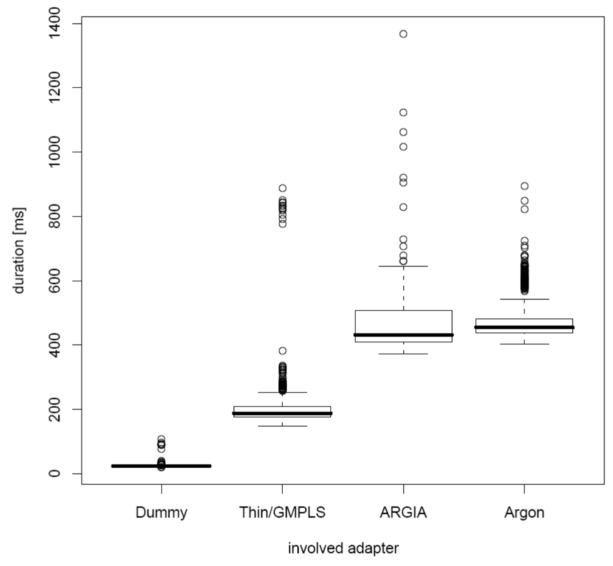

Figure 1.9: Request response time of each adapter located in the real test-bed

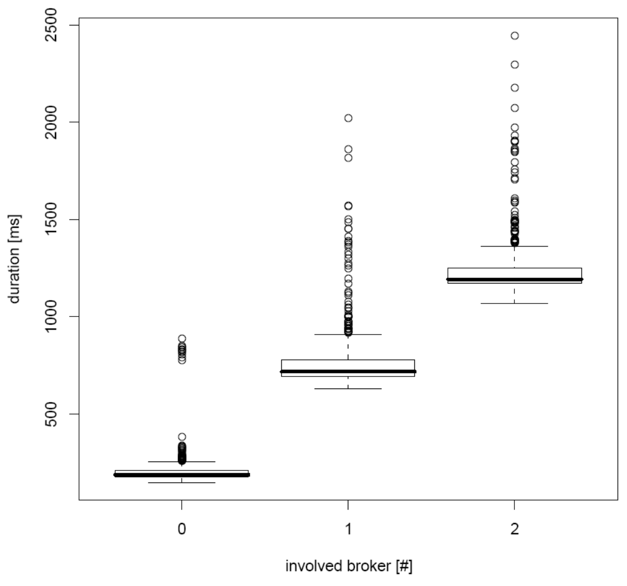

Figure 1.10: Time response when adding IDB in the hierarchy level

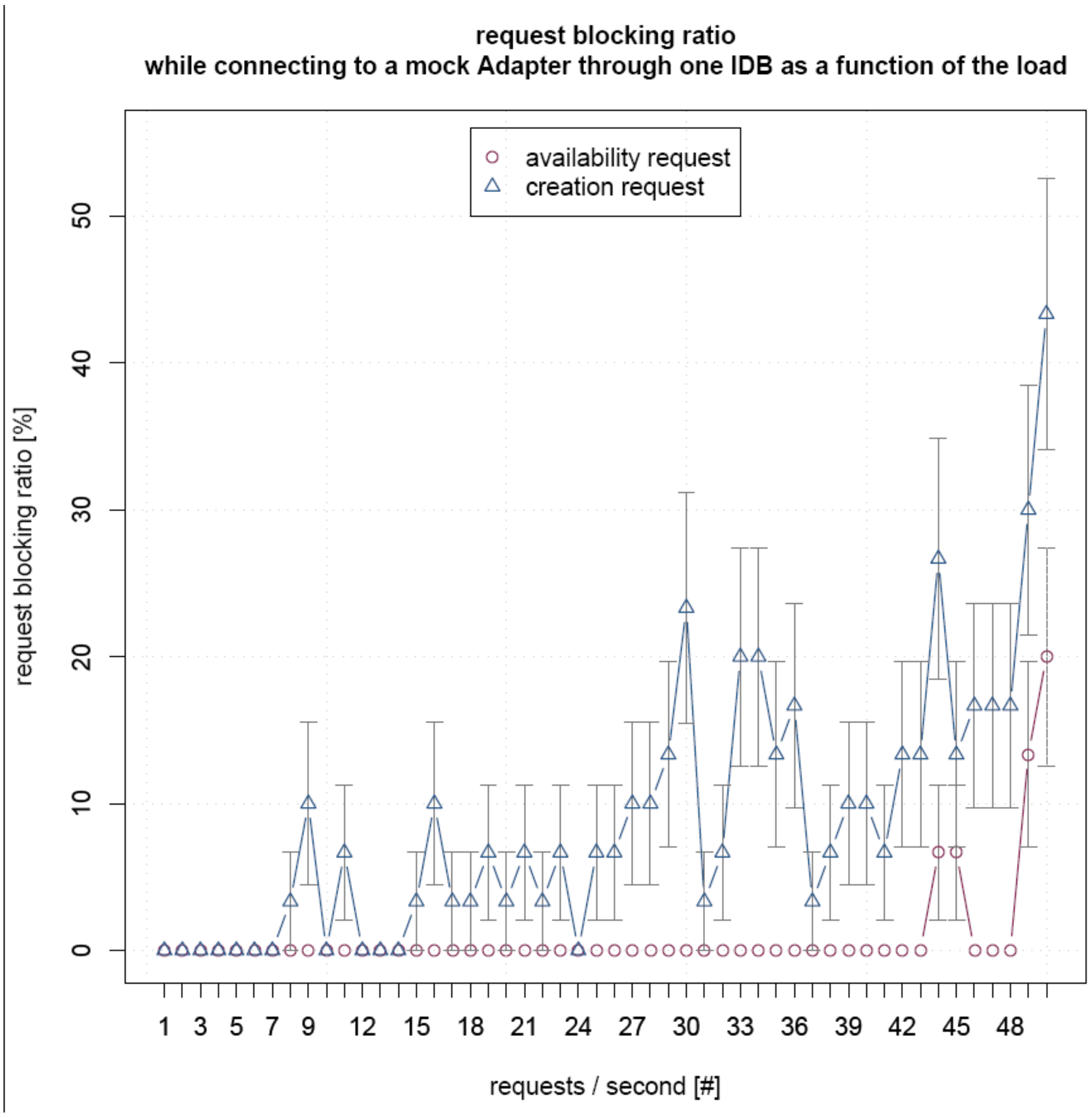

Figure 1.11: Request blocking ratio over one IDB

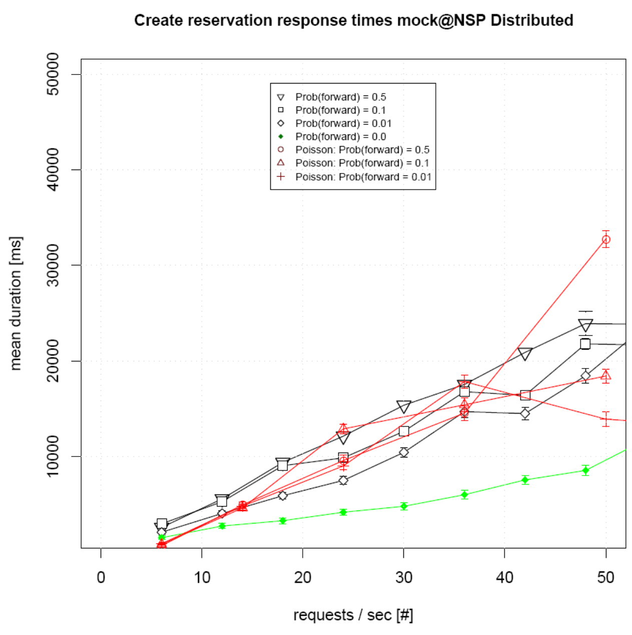

Figure 1.12: Distributed NSP time response under distinct job inter-arrival times

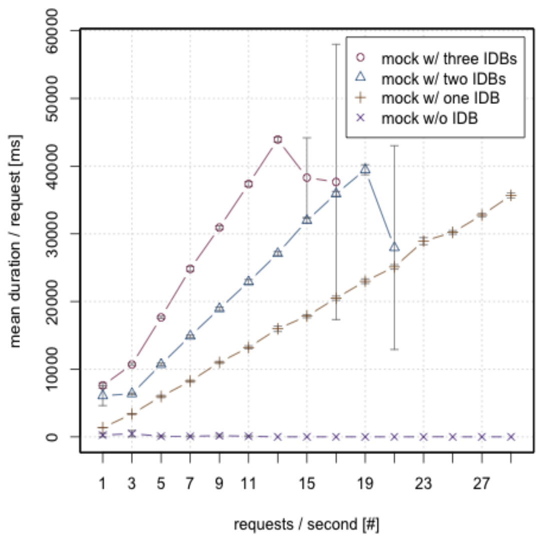

Figure 1.13: Request response time when connecting to a mock Adapter through different layers of IDBs

1.4.3 Analysis based on the figures 1.8, 1.11, 1.12, and 1.13

To evaluate the performance and scalability of the current service plane implementation, both centralized and distributed, the request response time and request blocking ratio as a function of the load were measured. The measurements were executed on a system based on two Intel Core 2 CPUs with 2.66 GHz each, and the population requests were generated from two PCs. The results show that the system is stable up to 50 requests per second; that means that the system is able to handle successfully all the reservation requests. To obtain statically reliable results, 30 repetitions were made and the error bar indicates on standard error of the mean.

Based on a uniform distribution or a Poisson distribution the number of requests per second was increased, and a timeout of 50 seconds between each IDB involved in the NSP and its corresponding emulated NRPS was introduced. Results confirm that as the probability of forwarding the request in one IDB increases, the time response of the whole service plane is affected, decreasing its performance. However, in comparison to the centralized approach, the results show that the distributed NSP can handle successfully more number of simultaneous requests.

In this context, a further issue influences the design of the service plane topology. Different requests and response types need specific bandwidths. Based on measurements in the PHOSPHORUS test-bed, next table shows the average size of the four most used requests. Given these values, it can be stated that a complete reservation workflow (involving checking availability, setting up a connection, and cancelling it after use) requires about 13kB per reservation between two involved entities. The topology updates were sent every five minutes and therefore just consumed about 18 bytes/s., again only between two entities. That implies that at a load of 30 req/s connections with at least 380 kb/s are required between the IDBs and Adapters.

Description	Requests [bytes]	Response [bytes]	∑ [bytes]
Availability	2768	2072	4840
Creation	2792	1966	4758
Cancellation	1572	1786	3358
Topology Updates	3539	1760	5299

Table 1.6: Average size of different request and response messages

1.5 Test conclusions

Harmony is a system that is able to accept multiple simultaneous provisioning requests directly from the GRID middleware under a centralized or a distributed approach. According to the number of the requests that can be successfully processed, the results obtained in the preceding tests indicate the best setup for distributed Network Service Plane of the Harmony system, deployed over the PHOSPHORUS test-beds, by validating the behaviour of the different emulated modes. So, considering the results we end up that a distributed approach is the most suitable for Harmony, since it improves the responsiveness behaviour by keeping down the number of involved IDB layers and by preventing an overload of single entities, making the system more efficient and scalable.

However, we also have to take into account that when the probability of forwarding the requests in one IDB increases, the time response of the whole service plane is affected. As distributed systems increase the load traffic due to the messaging, is interesting to point out that under maximum stress conditions considered for the system (30 requests/s connections) it has a maximum load of 380 kb/s between IDBs and the NRPSs.

Future work will validate the results obtained in a wider simulation scenario, an experimental network where real scalability will be deeply assessed. For that purpose the EU FP7 FEDERICA project experimental infrastructure will be used.

2 Control plane experiments – WP2

This section provides information about functional testing on the different modules implemented for G^{2}MPLS Control Plane. It reports on the specific functional tests conducted to identify and confirms the proper operation of the various G^{2}MPLS modules on real equipments deployed in some PHOSPHORUS test-beds.

2.1 Testing environment

This section reports the details of the PSNC and UESSEX test-beds used for the G^{2}MPLS Control Plane test. One fibre switched test-bed from UESSEX, one wavelength switched test-bed from PSNC and two Ethernet switched test-beds were involved in the test. **Table 2.1** shows equipment inventory in PSNC and UESSEX local test-beds involved in the testing.

EQUIPMENT					
PSNC			UESSEX		
Type/Make/ Model	No.	Attrib.	Type Make/Model	No.	Attrib.
ADVA FSP 3000RE-II (Lambda Switch)	3	15 Pass through ports 6 Local ports 3 lambda configured	CalientDiamondWave (Fibre Switch)	1 (4 after partitioning)	96 ports total
VLAN capable GE switch (XMR 8000/ MLX-16 Foundry)	3 (5 after partitioning)	120 ports	VLAN capable GE switch (FastIronFoundry)	1	24 ports optical
Allied Telesis AT-8000S	1 (3 after partitioning)	48 x 10/100 ports 2 x 1000 ports	VPN Gateway	1	----
Allied Telesis AT-9424T	1	24 x 10/100/1000 ports	Equipment Controller	4	PCs
Equipment controller	4	Intel Pentium 4, 2 x Intel XEON E5405, Fujitsu-Siemens PRIMERGY RX200	Client nodes	3	Intel dual core processor servers
Client nodes	2	HP IA64 2xIntel Itanium2 servers			

Table 2.1: Summary of available equipments in PSNC and UESSEX domains for G^{2}MPLS functional tests

2.2 G^{2}MPLS Control Plane functional tests

This section updates the previous release of deliverable D6.8 with new functionality test-cards. The tests are related to the following functionalities:

- route calculation for Call and LSPs,
- call and LSP signalling in multi-domain and multi-technology networks,
- routing and signalling resiliency.

G^{2}MPLS Control Plane functionality	Status
Advertisement of Network topology and availabilities	Available
Discovery and advertisement of GRID resources capabilities and availabilities	Available
Multi-domain resources advertisement abstractions	Available
Routing resiliency	Available
Optical network impairments	Available
Single-step service setup and teardown	Available
Multi-domain path/routing computation	Available
Single-domain and multi-domain service setup	Available
Advanced and on-time reservation for Network resources	Available
Advanced and on-time reservation for GRID resources	Available
Co-allocation of Network&GRID resources during service setup	Available
Service resiliency	Available
Uni-directional or bi-directional point-to-point LSP connections	Available
Anycast point-to-point connections	Available
Flexible bandwidth allocation	Available
Multi-technology stitching (FSC, LSC, ETH, etc)	Available
Interworking with legacy Transport Network equipment	Available
Management interfaces	Available

Table 2.2: G^{2}MPLS Control Plane functionalities available for testing

2.2.1 Overview of the tests

The functionality of the G^2MPLS Control Plane was tested using 33 test-cards divided into two main areas. Each test-card verifies a set of G^2MPLS features expressed as a set of objectives to be achieved in fixed environment conditions. All the test-cards have been successfully executed and thus the current state of the G^2MPLS Control Plane prototype development fulfils the expected functionalities as per Milestone M2.8.

Route computation tests			
	Route computation in LSC domain environment by G^2.PCE module		
No	**Test Card**	**Test name**	**Status**
1	G^2MPLS-TC-7.1	G^2.PCE retrieves from routing the topology information of the LSC domain	Passed
2	G^2MPLS-TC-7.2	Notification about topology change in LSC domain	Passed
3	G^2MPLS-TC-7.3	Route Computing in LSC domain between ingress and egress TNAs endpoints	Passed
4	G^2MPLS-TC-7.4	Route Computing in LSC domain between ingress TNA endpoint and egress GRID endpoint	Passed
	Route computation in FSC domain environment by G^2.PCE module		
5	G^2MPLS-TC-8.1	G^2.PCE retrieves from routing the topology information of the FSC domain	Passed
6	G^2MPLS-TC-8.2	Notification about topology change in FSC domain	Passed
7	G^2MPLS-TC-8.3	Route Computing in FSC domain between ingress and egress TNAs endpoints	Passed
8	G^2MPLS-TC-8.4	Route Computing in FSC domain between ingress TNA endpoint and egress GRID endpoint	Passed
	Route computation in multi-domain environment by G^2.PCE module		
No	**Test Card**	**Test name**	**Status**
9	G^2MPLS-TC-9.1	G^2.PCE retrieves from routing the topology information of both domains	Passed
10	G^2MPLS-TC-9.2	Route Computing through both domains between ingress and egress TNAs endpoints	Passed
11	G^2MPLS-TC-9.3	Route Computing through both domains between ingress TNA endpoint and egress GRID endpoint	Passed
G^2MPLS signalling tests			
	Multi-domain call signalling tests in case of LSC-FSC technology stitching		
No	**Test Card**	**Test name**	**Status**
12	G^2MPLS-TC-10.1	Setup of one bidirectional path starting in LSC domain and ending in FSC domain	Passed
13	G^2MPLS-TC-10.2	Teardown of the one bidirectional path starting in LSC domain and ending in FSC domain	Passed
14	G^2MPLS-TC-10.3	Setup of one bidirectional path starting in FSC domain and ending in LSC domain	Passed
15	G^2MPLS-TC-10.4	Teardown of the one bidirectional path starting in FSC domain and ending in LSC domain	Passed
	Signalling tests in ETH domain		
No	**Test Card**	**Test name**	**Status**
16	G^2MPLS-TC-11.1	ETH node initialization	Passed
17	G^2MPLS-TC-	Transport Plane notifications from ETH node	Passed

No	Test Card	Test name	Status
	11.2		
18	G^{2}MPLS-TC-11.3	Setup of one bidirectional ETH LSP	Passed
19	G^{2}MPLS-TC-11.4	Tear down of one bidirectional ETH LSP from HEAD node	Passed
20	G^{2}MPLS-TC-11.5	Tear down of one bidirectional ETH LSP from TAIL node	Passed
21	G^{2}MPLS-TC-11.6	Unsuccessful bidirectional ETH LSP setup (failure in HEAD node)	Passed
22	G^{2}MPLS-TC-11.7	Unsuccessful bidirectional ETH LSP setup (failure in intermediate node)	Passed
23	G^{2}MPLS-TC-11.8	Unsuccessful bidirectional ETH LSP setup (failure in TAIL node)	Passed
24	G^{2}MPLS-TC-11.9	Setup of one bidirectional ETH LSP with advance reservation	Passed
25	G^{2}MPLS-TC-11.10	Tear down of one bidirectional ETH LSP with advance reservation from HEAD node	Passed
G^{2}MPLS resiliency tests			
	Routing resiliency		
No	**Test Card**	**Test name**	**Status**
26	G^{2}MPLS-TC-12.1	Restart of the G^{2}MPLS client controller	Passed
27	G^{2}MPLS-TC-12.2	Restart of the G^{2}MPLS core controller	Passed
28	G^{2}MPLS-TC-12.3	Restart of the G^{2}MPLS egde controller	Passed
29	G^{2}MPLS-TC-12.4	Restart of the G^{2}MPLS border controller	Passed
	Signalling resiliency		
No	**Test Card**	**Test name**	**Status**
30	G^{2}MPLS-TC-13.1	Restart of the G^{2}MPLS client controller	Passed
31	G^{2}MPLS-TC-13.2	Restart of the G^{2}MPLS core controller	Passed
32	G^{2}MPLS-TC-13.3	Restart of the G^{2}MPLS egde controller	Passed
33	G^{2}MPLS-TC-13.4	Restart of the G^{2}MPLS border controller	Passed

Table 2.3: Overview of the executed test-cards

2.2.2 Route computation tests

The G^2.PCE module is responsible for computation of the routes based on topology information retrieved from the routing protocol (G^2.OSPF-TE). The routes calculation is done in two cases:

- G^2.NCC request call ERO, and
- G^2.RC request LSP ERO.

The call ERO and LSP ERO are needed for the call and the LSP signalling process.

Proper work of the G^2.PCE module has been tested in various Control Plane and Data Plane configurations. The G^{2}MPLS route computation tests have been executed in three separate sessions:

- Route computation in the LSC domain
- Route computation in the FSC domain
- Route computation in the multi-domain

Proper work of the G^2.PCE module was verified during network and GRID topology information retrieving from the G^2.OSPF-TE module and during call and LSP ERO computations.
Route computations were performed in the LSC domain topology shown in **Figure 2.1** and the FSC domain topology shown in **Figure 2.2**.
Route computation in multi-domain environment tests verified the call ERO computation in case of endpoints located in two different domains (see used logical topology in **Figure 2.3**).

All tests were repeated for different egress endpoint specification:

- egress TNA specification (G^{2}MPLS unicast),
- GRID requirement specification for egress User Network Interface – Client side UNI-C (G^{2}MPLS anycast).

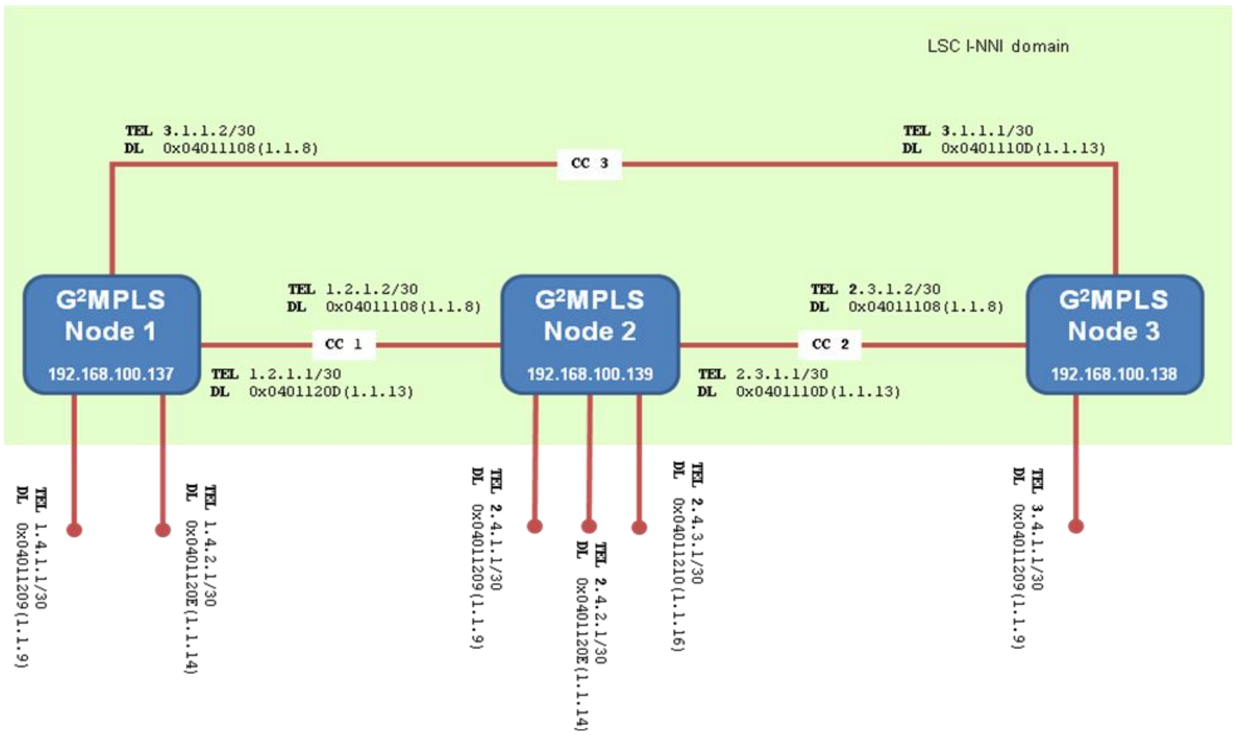

Figure 2.1: Logical topology of the single-domain LSC test-bed

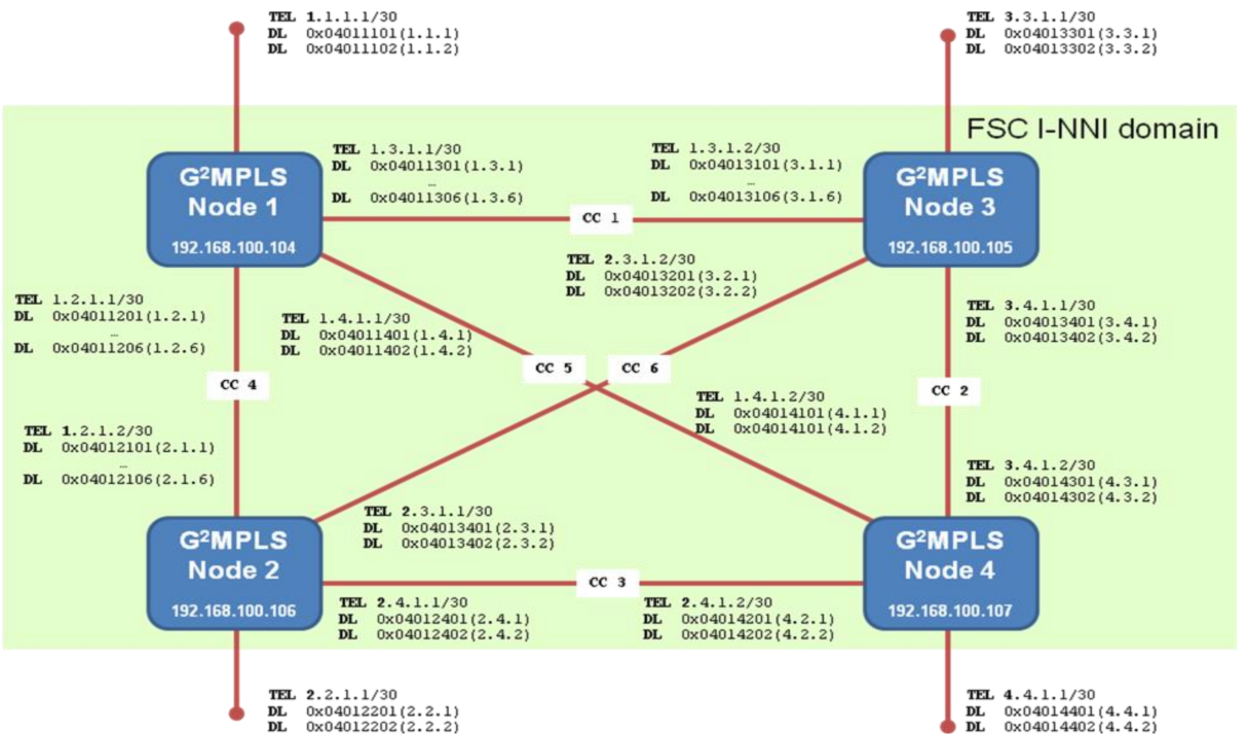

Figure 2.2: Logical topology of the single-domain FSC test-bed

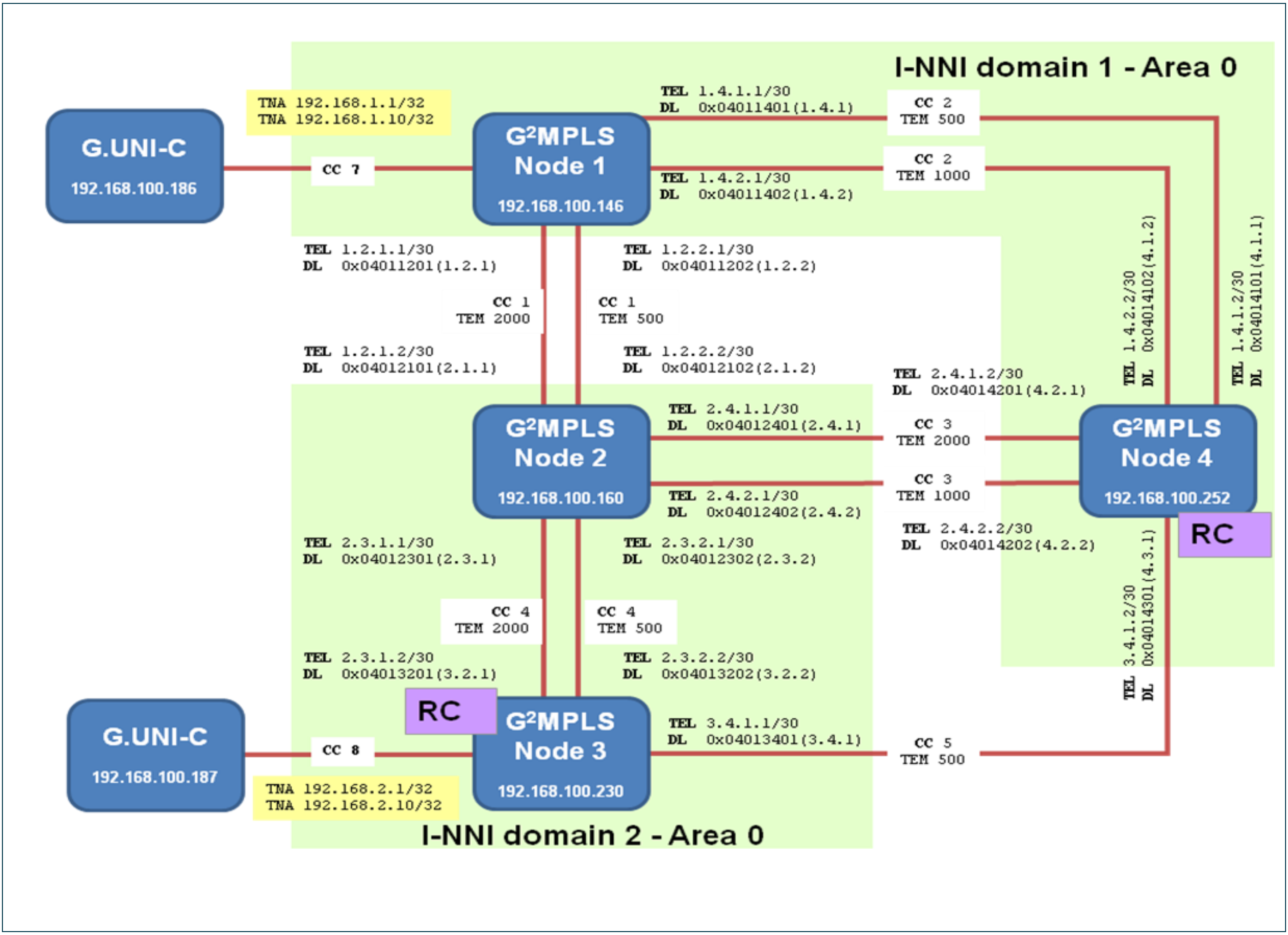
I-NNI domain 1 - Area 0
I-NNI domain 2 - Area 0
G.UNI-C 192.168.100.186
G.UNI-C 192.168.100.187
G²MPLS Node 1 192.168.100.146
G²MPLS Node 2 192.168.100.160
G²MPLS Node 3 192.168.100.230
G²MPLS Node 4 192.168.100.252
RC
RC
TNA 192.168.1.1/32
TNA 192.168.1.10/32
TNA 192.168.2.1/32
TNA 192.168.2.10/32
CC 7
CC 8
TEL 1.4.1.1/30 DL 0x04011401(1.4.1)
TEL 1.4.2.1/30 DL 0x04011402(1.4.2)
CC 2 TEM 500
CC 2 TEM 1000
TEL 1.2.1.1/30 DL 0x04011201(1.2.1)
TEL 1.2.2.1/30 DL 0x04011202(1.2.2)
CC 1 TEM 2000
CC 1 TEM 500
TEL 1.2.1.2/30 DL 0x04012101(2.1.1)
TEL 1.2.2.2/30 DL 0x04012102(2.1.2)
TEL 2.4.1.2/30 DL 0x04014201(4.2.1)
TEL 1.4.2.2/30 DL 0x04014102(4.1.2)
TEL 1.4.1.2/30 DL 0x04014101(4.1.1)
TEL 2.4.1.1/30 DL 0x04012401(2.4.1)
TEL 2.4.2.1/30 DL 0x04012402(2.4.2)
CC 3 TEM 2000
CC 3 TEM 1000
TEL 2.4.2.2/30 DL 0x04014202(4.2.2)
TEL 2.3.1.1/30 DL 0x04012301(2.3.1)
TEL 2.3.2.1/30 DL 0x04012302(2.3.2)
CC 4 TEM 2000
CC 4 TEM 500
TEL 2.3.1.2/30 DL 0x04013201(3.2.1)
TEL 2.3.2.2/30 DL 0x04013202(3.2.2)
TEL 3.4.1.2/30 DL 0x04014301(4.3.1)
TEL 3.4.1.1/30 DL 0x04013401(3.4.1)
CC 5 TEM 500

Figure 2.3: Logical topology of the multi-domain test-bed.

2.2.3 G^{2}MPLS signalling tests

The signalling purpose of G^{2}MPLS is to establish and destroy the call and the LSP in the G^{2}MPLS Control Plane. The call and LSP setup and teardown operations are managed by G^2.NCC and G^2.RC modules.

The G^{2}MPLS signalling tests have been executed in two separate sessions:

- multi-domain signalling tests in case of technology stitching
- single-domain signalling tests in case of the Ethernet domain

The multi-domain tests have been used to verify the proper work and interaction of modules involved in the multi-domain signalling (mainly G^2.NCC, G^2.RC, G^2.PCE and G^2.ENNI-RSVP). During these tests, ingress and egress TNA were located in different domains.

Figure 2.4 shows the multi-domain call signalling test-bed. In the test-bed there were two domains. The PSNC LSC domain was composed of 4 different G^2MPLS controllers:

- one G^2MPLS core controller,
- one G^2MPLS edge controller,
- one G^2MPLS G.UNI client controller,
- one G^2MPLS border controller (with inter-domain Routing Controller).

UESSEX FSC domain contained 7 different G^2MPLS controllers:

- three G^2MPLS edge controllers,
- three G^2MPLS G.UNI client controllers,
- one G^2MPLS border controller (with inter-domain Routing Controller).

The calls and LSPs were established between all pairs of clients; between PSNC client and each UESSEX client and also between each pair of the UESSEX clients.

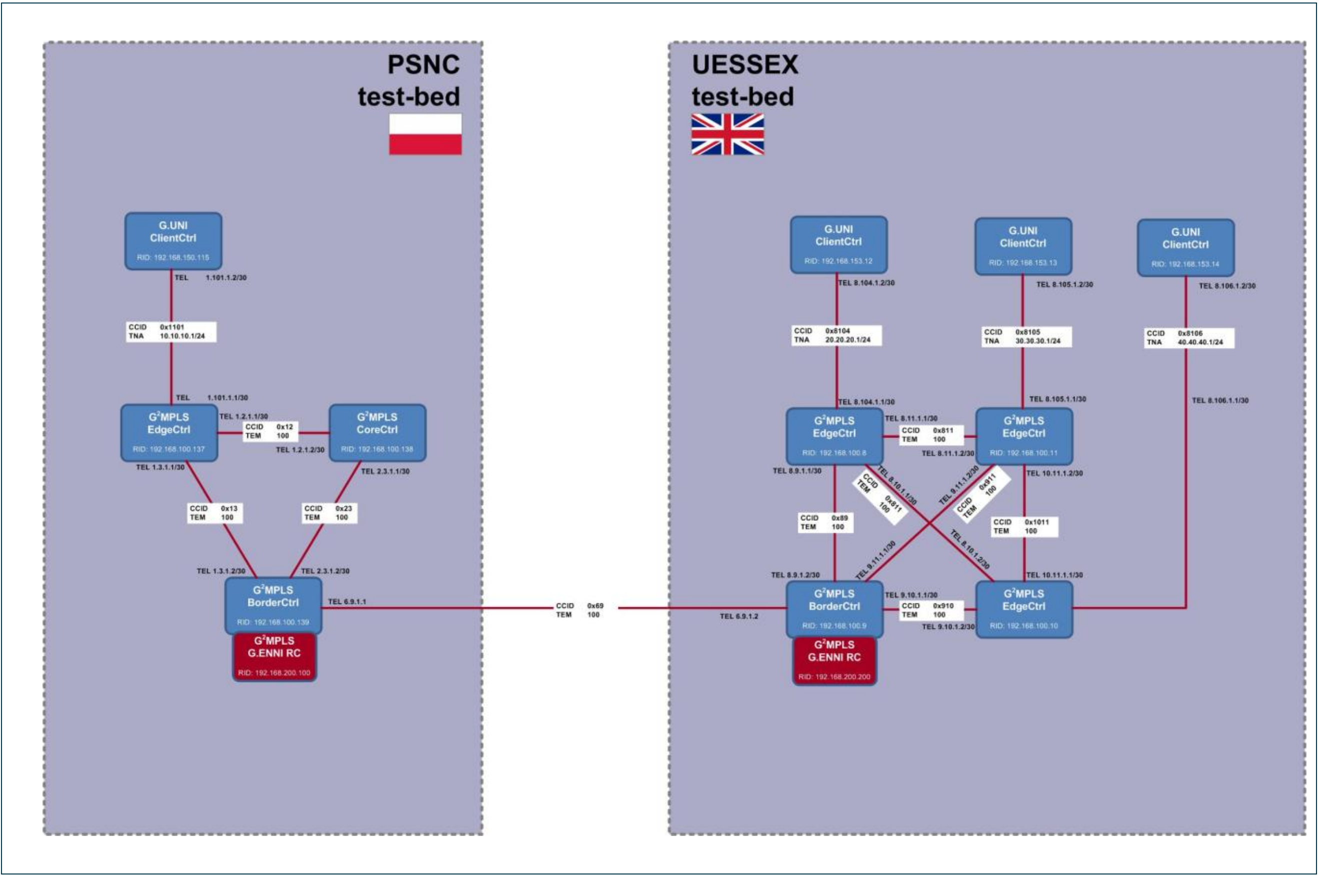

Figure 2.4: Logical topology of the multi-domain LSC and FSC test-bed for G^2MPLS call signalling tests

During the tests it was possible to trace the communication between different modules of each controller participating in signalling. For example the call signalling in relation of G^2.CCC, G^2.NCC and G^2.PCE modules is presented in **Figure 2.5**. The characteristic was that only ingress G^2.NCC queries for call route through all domains (call ERO). The G^2.UNI-RSVP and G^2.ENNI-RSVP are not presented in the picture but they also took part in the call signalling process. **Figure 2.6** presents LSC signalling in the area of one domain. G^2.NCC sends a request to G^2.RC for LSP creation, then G^2.RC asks G^2.PCE for LSP route (LSP ERO) and sends the request to the ingress G^2.INNI-RSVP module. The egress G^2.INNI-RSVP module sends notification to G^2.RC and G^2.NCC.

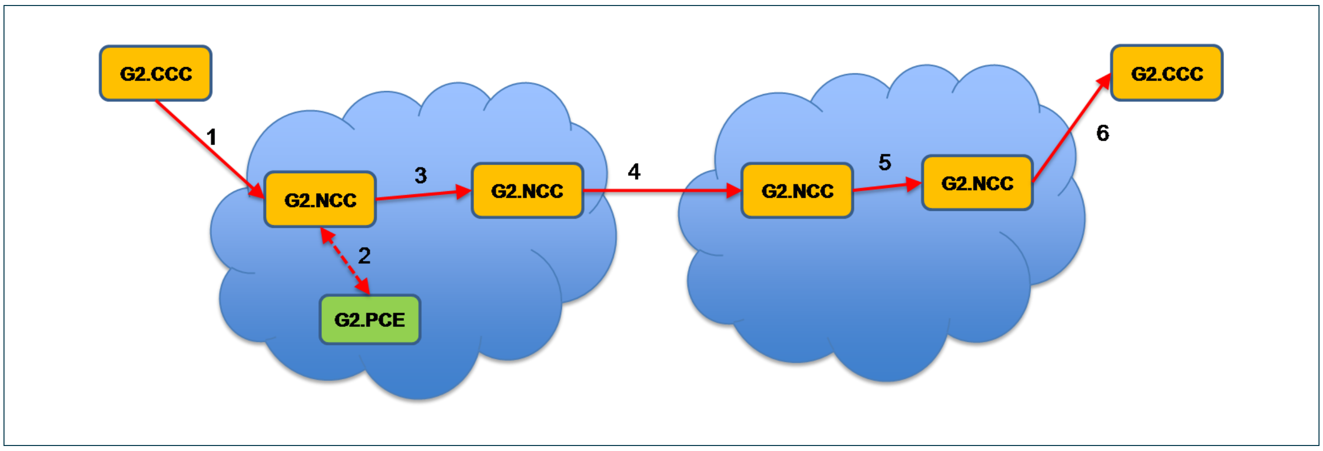

Figure 2.5: Chain of actions traced during inter-domain call signalling tests.

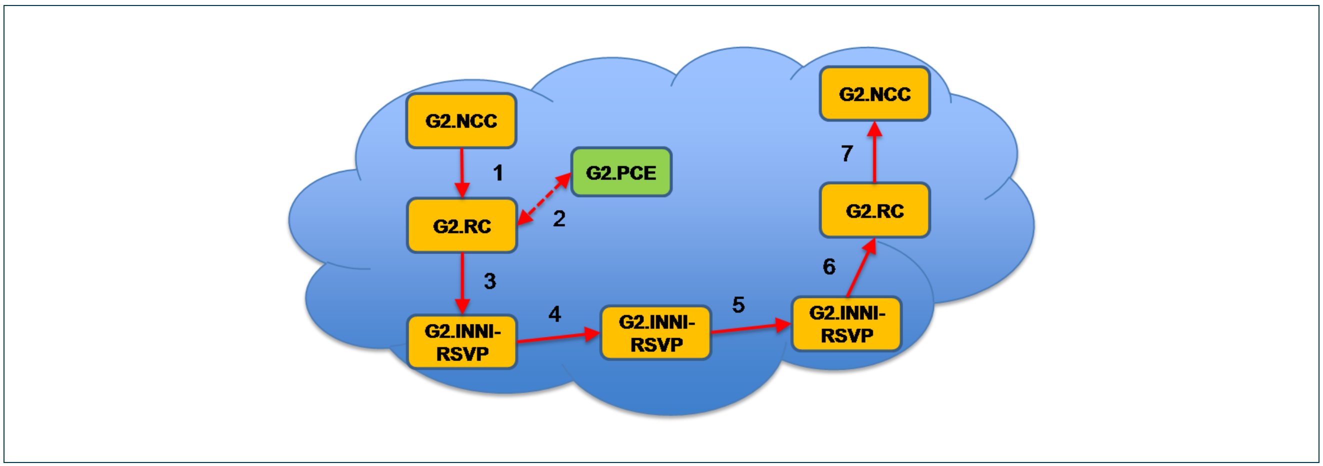

Figure 2.6: Chain of actions traced during intra-domain LSP signalling tests.

Additionally, the single-domain signalling tests were performed in the Ethernet domain. These tests were performed in a similar way as the related tests for FSC and LSC domains.

In **Figure 2.7** the logical topology of the test-bed for the Ethernet domain call signalling tests is described. For the purpose of the tests, 6 G^{2}MPLS controllers were deployed. The tests were repeated a few times, using the same logical topology, on the following types of devices installed in the Data Plane:

- Foundry XMR-8000,
- Foundry MLX-16,
- cheap Allied Telesis switches (AT-8000S, AT-9492T).

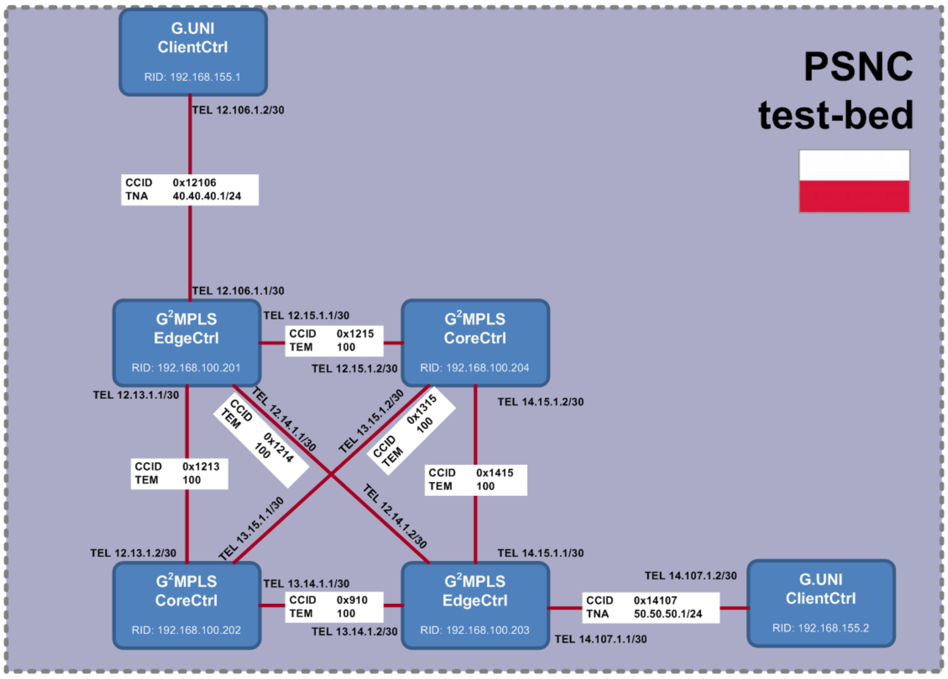

Figure 2.7: Logical topology of the Ethernet domain test-bed for G^{2}MPLS Call signalling tests

2.2.4 G^{2}MPLS resiliency

G^{2}MPLS resiliency guarantees the restoration of the G^{2}MPLS controller proper state in case of faulty conditions or the controller restarting. It means that the routing state and signalling state will be restored, and the G^{2}MPLS controller will be able to continue the work without affecting the running services.

The G^{2}MPLS resiliency tests have been executed in two separate sessions:

- Routing resiliency, and

- Call and LSP signalling resiliency.

The routing resiliency tests were used to verify the proper work of routing modules: G^2.UNI-OSPF, G^2.INNI-OSPF, G^2.ENNI-OSPF in case of restarting one of the G^2MPLS controllers in the Control Plane. The restarted controller can belong to one of the following types:

- G^2MPLS client controller (G^2.UNI-OSPF),
- G^2MPLS edge controller (G^2.INNI-OSPF and G^2.UNI-OSPF),
- G^2MPLS core controller (G^2.INNI-OSPF),
- G^2MPLS border controller (G^2.INNI-OSPF and G^2.ENNI-OSPF).

After the controller is restarted the information about network topology and GRID resources stored in OSPF databases must be synchronized with information available in the routing domain (in peer routing controllers) in a few seconds.

The routing resiliency tests were performed in the single-domain testbed (**Figure 2.7**) and in the multi-domain testbeds (**Figure 2.3** and **Figure 2.4**).

The Call signalling resiliency tests were used to verify the proper work of routing modules: G^2.CCC, G^2.NCC and G^2.RC. The states of each module are stored in a backup file and the controller retrieves the previous state from the backup file during the restart procedure.

The LSP signalling resiliency tests were used to verify the proper work of routing modules: G^2.PC, G^2.INNI-RSVP and TNRC in case of restarting the controllers currently serving LSPs. In case of restarting the G^2MPLS controller, the G^2.PC retrieves information about all installed LSPs from a backup file and requests restoration to the G^2.INNI-RSVP module. The G^2.INNI-RSVP module requests restoration of state in the TNRC module. The TNRC module checks if the proper cross-connections are installed in the Transport Network equipment.

The signalling resiliency tests were performed in the single-domain testbed (**Figure 2.7**) and in the multi-domain testbeds (**Figure 2.3** and **Figure 2.4**).

2.2.5 Demonstrations of the G^2MPLS core functionalities

A demonstration was prepared in order to present the most interesting G^2MPLS functionalities. The demonstration overview is shown in **Figure 2.8**. The demonstration was performed at three conferences events: Supercomputing’08, ICT’08 and TNC’09.

The demonstration presents the G^2MPLS Control Plane in the multi-domain and the multi-technology testbed serving requests of the Distributed Data Storage System (DDSS) applications. The G^2MPLS testbed is composed of two network domains:

- the LSC domain containing three ROADM ADVA FSP 3000RE-II devices in PSNC (Poland), and

- the FSC domain with four independent virtualized nodes based on the Calient DiamondWave FiberConnect optical switch located in the University of Essex laboratory (United Kingdom).

Both domains are interconnected using the 1Gbit/s GÉANT2 data plane link. The Control Plane inter-domain connectivity is available in the form of a G.E-NNI reference point.

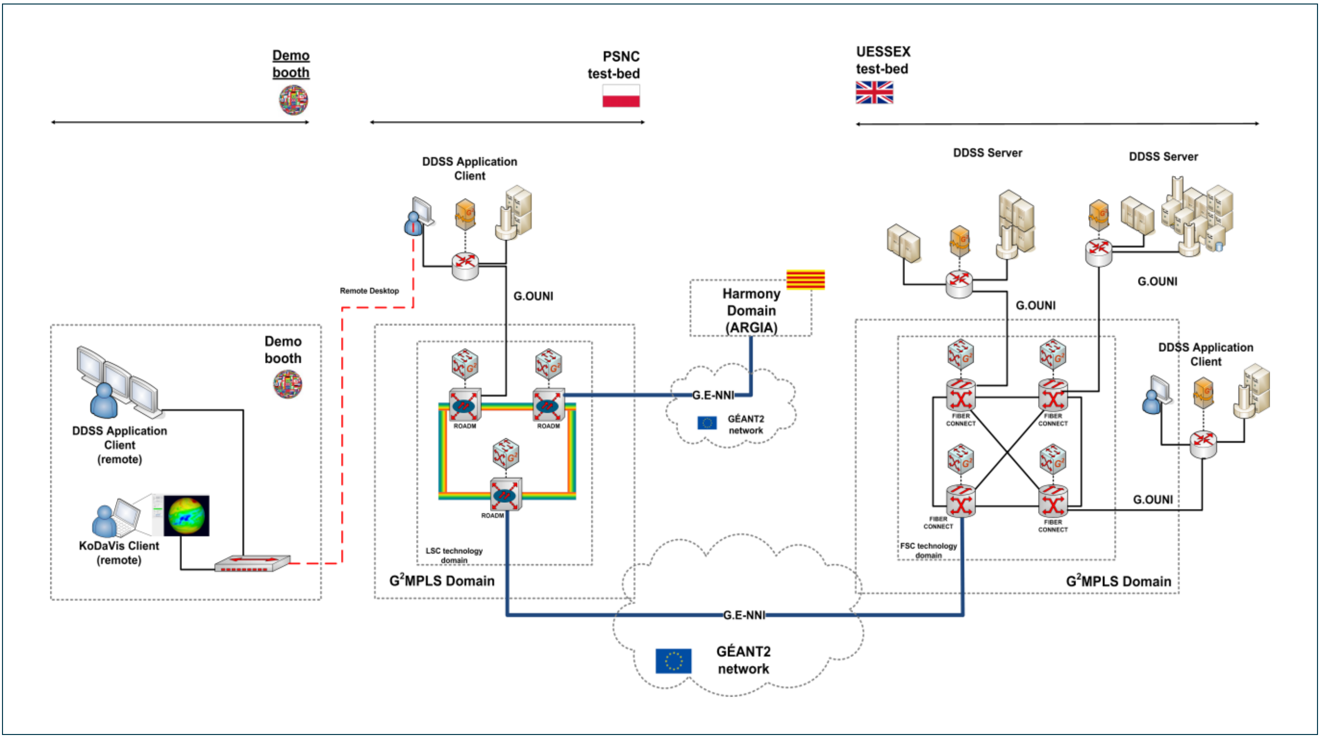

Figure 2.8: The SC'08, ICT'08 and TNC'09 demonstration testbed: Multi-domain G^{2}MPLS Control Plane.

The DDSS client application, remotely connected to the demonstration booth, was located in PSNC and connected to the network domain through the G.OUNI interface. This DDSS application offers the large files backup service using GRID-FTP (part of the Globus toolkit) and free storage spaces located in the DDSS server nodes.

When the user chooses to backup a big file from his/her local file system, the DDSS application sends an anycast request specifying the size of the file to the G^{2}MPLS Control Plane. The G^{2}MPLS edge node chooses the best suitable DDSS server with enough free storage space to handle the request, and proceeds with a setup of the high-bandwidth transaction over the two network domains. The setup and teardown of the path is shown on visualization of the Control Plane webpage.

The demonstration presents the following G^{2}MPLS functionalities:

- discovery of GRID resources,
- multi-domain network and GRID resource availability advertisement,
- anycast (and unicast) point-to-point service setup and teardown,
- multi-domain path computation,
- LSP signalling and Transport Network equipment configuration (lambda and fiber switching),
- Management interfaces.

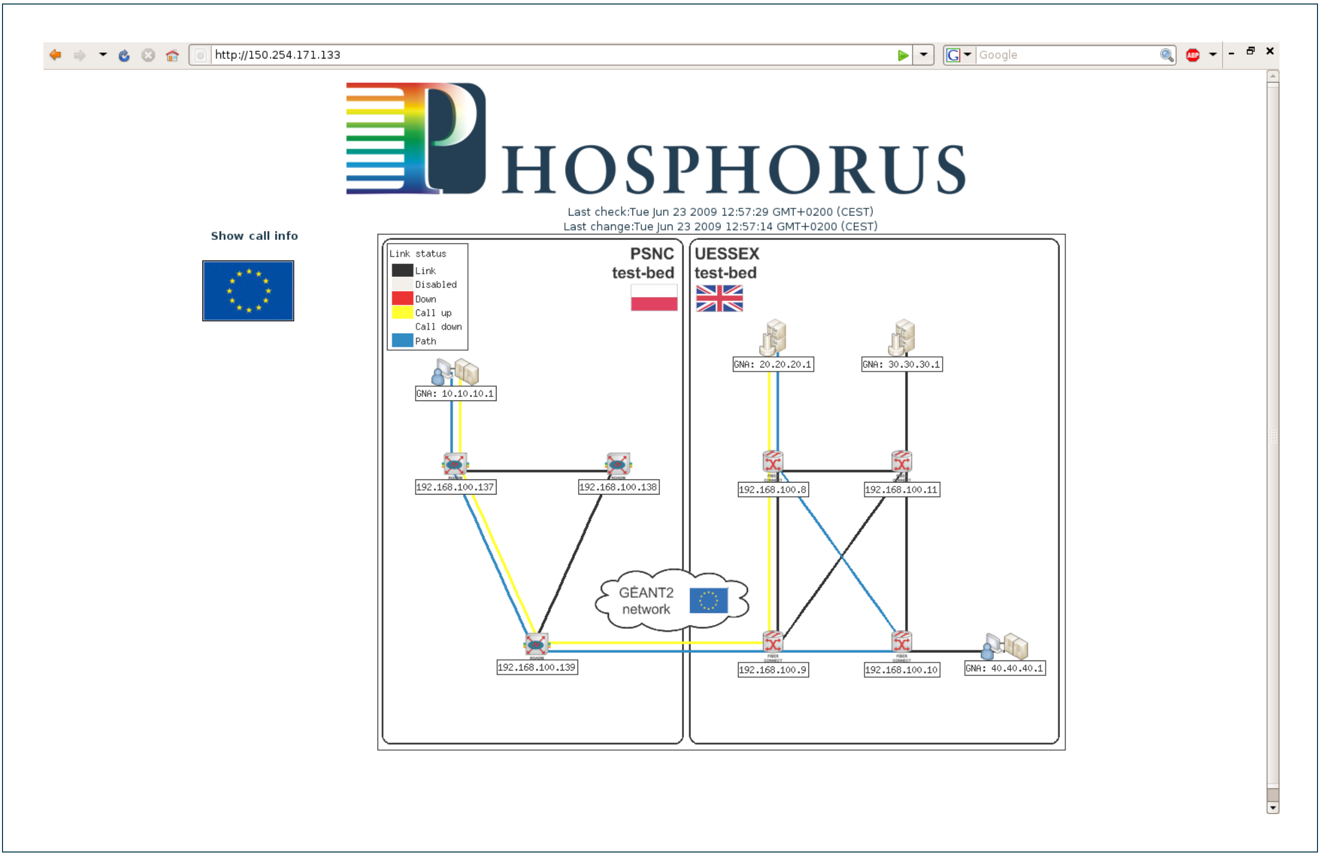

Figure 2.9: The visualization of the G^{2}MPLS Control Plane demonstration testbed.

2.3 Test conclusions

The G^{2}MPLS Control Plane, created as the main development of the PHOSPHORUS WP2 team, meets expectations presented as a set of requirements in deliverables D2.1 [PHOSPHORUS-D2.1] and D2.6 [PHOSPHORUS-2.6]. The G^{2}MPLS CP functionalities were well tested during functionality acceptance tests, during periodical tests and also before live demonstrations. The G^{2}MPLS Control Plane currently contains both pure GMPLS functionalities and GRID-enabled functionalities. The G^{2}MPLS Control Plane was tested in various environment settings: LSC, FSC and Ethernet Transport Plane technologies, single and multi-domain, technologies stitching. The most advanced and innovative functionality available in G^{2}MPLS is the Anycast service request and co-allocation of Network and GRID resources in response to the request.

3 Middleware and applications experiments – WP3

The following section describes the experiments carried out with middleware and applications within WP3 during the last period of the PHOSPHORUS project.

The middleware experiments focussed on both the MSS/UNICORE environment and INCA, the Intelligent Network Caching Architecture (details on the middleware can be found in deliverable D3.2 [PHOSPHORUS D3.2]. The goal was to provide the middleware support necessary for the final experiments and demonstrations of the applications. On the side of the applications experiments and demonstrations were conducted with Collaborative Data Visualization (KoDaVIS), Wide In Silico Docking On Malaria (WISDOM) and Distributed Data Storage Systems (DDSS). A detailed description of these applications can be found in deliverable D3.3 [PHOSPHORUS-D3.3].

3.1 Middleware experiments

Most WP3 middleware experiments in this phase were involving WP1 and WP2. WP3 internal experiments related to the preparation of demonstrations and the effects of integrating with HARMONY and G^{2}MPLS. The experiments of INCA focussed on improvements of performance.

3.1.1 MetaScheduling Service (MSS)

The MSS was in productive use in the testbed for quite some time. However, the latest G^{2}MPLS developments of WP2 and the final integration with HARMONY – including the seamless security chain from the middleware layer to the network layer – could not be completed in the previous phase of the PHOSPHORUS project but was done in this last period. Thus, the WP3 internal experiments with the MSS were centred around the integration with G^{2}MPLS and HARMONY. The full integration with G^{2}MPLS could already be demonstrated with the KoDaVis application during the PHOSPHORUS workshop at the TNC in Malaga. The full integration with HARMONY is presented with the WISDOM application as a use-case.

3.1.2 UNICORE

The UNICORE overall middleware environment as one of the foundations of the PHOSPHORUS testbed was fully functional and stable already in the previous periods. Only minor adjustments were necessary for the final period to support the planned experiments and demonstrations as described below.

In order to fully support all scenarios of the inter-WP demonstrations at SC08 and ICT2008, the UNICORE client plugin for the KoDaVis application was extended to select the desired demonstration scenarios. Additional changes were necessary in the negotiation protocol, to make the reservation of bandwidth as transparent to the application as possible. Data for the KoDaVis application can now be served by servers either selected by the client, the middleware (both unicast from the point of view of the network), or the network (anycast). From the point of view of the application and UNICORE client plugin, they all appear in the same way.

Demonstrations were given at the SC08 in Austin, Texas, the ICT2008 in Lyon, France, and the TNC2009 in Málaga, Spain. For the demonstrations at TNC2009, a local, transportable testbed has been installed, which contained all PHOSPHORUS components in a few machines.

3.1.3 INCA

During the final period - as we had already clarified the architectural decisions of INCA in the previous periods - we focused on algorithms and parameters to optimize the performance of INCA. The technical objectives for the development of this period were[1]:

1. The definition of a routing protocol in Content Addressable Network (CAN) as the routing latency through it determines the speed in which we can put and retrieve files though a storage area network.
2. The creation of a mechanism where we can use and balance the distribution of objects through the storage area network according to the bandwidth and storage capabilities of the participating nodes.
3. The development of an algorithm where we fragment objects in a specific block size and we can transfer them efficiently with high bit-rate by exploiting the resources of the network efficiently.

3.1.3.1 *Results of the final period*

A. Routing protocol in CAN

We have experimented during this period with multiple routing protocols. The first was routing according to the proximity of the point in CAN where we want to route agnostic to the underlying network conditions. The second was a routing where each time the message is forwarded to the closest node in the underlying network that is towards the point that we want to route. Finally, we have concluded to a routing mechanism where we route

[1] See deliverable D3.3 [PHOSPHORUS-D3.2]

each message according to a metric that is correlated with the proximity to the underlying network measured by the RTT (round trip time between nodes) and the distance in the virtual space between the neighbour and the final destination.

B. According to the findings there are two techniques that can work orthogonal towards the balancing mechanism of the storage area network. The first is the creation of an overlay where each node has a portion of the space according to its bottleneck resource and the second is the creation of object IDs dynamically not according to a uniform hash function but according to the remain resources of the participating nodes.

C. At last the fragmentation of objects into small blocks and the development of a flow control protocol is a technique very efficient towards the balancing of the storage resources of the system and the exploitation of underlying network resources.

3.2 Application Experiments

3.2.1 KoDaVIS

3.2.1.1 *Preparation and implementation of the KoDaVis Demonstration at the TNC2009 in Málaga*

The KoDaVis application has been in a stable state for a long time (it was already demonstrated during the first annual review at PSNC). It has been demonstrated at various occasions. Minor adaptations were required to support the scenarios foreseen by WP2, where either the middleware or the network selects the KoDaVis data server. The actual change was necessary to continue to support collaboration among multiple KoDaVis clients, even though their data is served by different data servers.

The KoDaVis application (Collaborative Data Visualization) is a visualization tool for scientists to collaboratively visualize large data sets from geographically dispersed locations. Bandwidth reservation as provided by the PHOSPHORUS project, is one of the essentials to allow for a good experience of the visualization.

Data for the KoDaVis application can be served by servers either selected by the client, the middleware (both unicast from the point of view of the network), or the network (anycast). From the point of view of the application and UNICORE client plugin, they all appear in the same way.

Reservation of Bandwitdth can be done in one of three scenarios.

a) Unicast – The application (UNICORE client plugin for KoDaVis) chooses a data server and requests a reservation from the MSS,

b) Anycast Overlay – The MSS, using the information retrieved from the network, chooses a better KoDaVis server to handle the KoDaVis client request, and requests the unicast connection in the G2MPLS Control Plane,

c) Anycast Integrated – The MSS passes the KoDaVIS client request to the network and the G2MPLS Control Plane takes a decision about the KoDaVis server and proceeds with a setup of the path in the Transport Plane.

Below is a diagram of the setup. The machines had been configured prior to the demonstrations and brought along to the PHOSPHORUS booth at TNC2009. KoDaVis data servers were deployed on two machines (on the right). The client could then request connections to them using either of the three scenarios mentioned above.

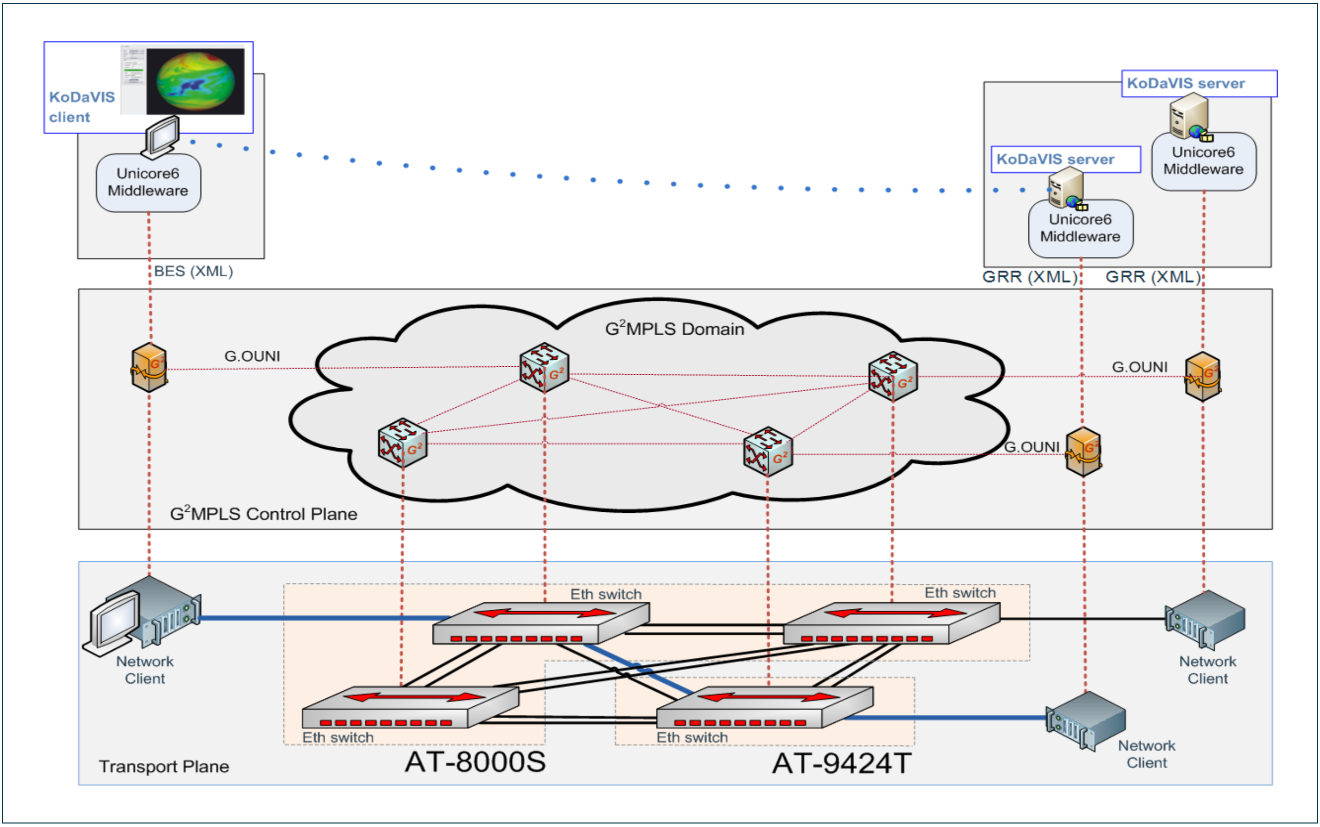

Figure 3.1: Setup of the KoDaVis demonstration using the WP2 testbed environment.

More details on the interoperation with WP2 can be found in Section 7: Integration between Control Plane and GRIDs and applications.

3.2.2 WISDOM

To enable the WISDOM application in the final testbed environment it was necessary to update the testbed with the latest versions of the MSS server, the WISDOM scheduler server and to apply a number of small modifications in the configuration files or the WISDOM application environment. The following tasks had been performed and the results had been successfully tested during the internal experiments:

A: Update MSS to Server version 0.1.1	(wsag4unicore6-agreement-factory-0.1.1)
B: Update WISDOM Scheduler Server version 1.0.3	(wisdom-scheduler-1.0.3)

3.2.2.1 *A: MSS Service Deployment*

Installation:

Download binary wsag4unicore6-distribution-0.1.1-server-bin.tar.gz from https://bpackcs-e0.scai.fraunhofer.de/wsag4unicore6 and deploy as an application in Tomcat

Required Changes in Configuration files:
1. Editing the required configuration file wsag4u6.config:
a. Adding UNICORE registries information.

```
<wsag4u6:Unicore6Configuration xmlns:wsag4u6="http://schemas.scai.fraunhofer.de/wsag4unicore6/">
   <wsag4u6:Registry>
<wsag4u6:URL>https://bpackcs-
e0.scai.fraunhofer.de:443/UNICORE6TEST/services/Registry?res=default_registry</wsag4u6:URL>
  <wsag4u6:Description>PACKCS Default Registry</wsag4u6:Description>
  </wsag4u6:Registry>
   <wsag4u6:Registry>
  <wsag4u6:URL>https://juggle-phos.fz-
juelich.de:8090/Phos_Registry/services/Registry?res=default_registry</wsag4u6:URL>
          <wsag4u6:Description>PHOSPHORUS Default Registry</wsag4u6:Description>
<wsag4u6:Excludes>
<wsag4u6:TSSName>node01.phosphorus.man.poznan.pl:6060/PSNC-SITE</wsag4u6:TSSName>
</wsag4u6:Excludes>
</wsag4u6:Registry>
```

2. Editing required configuration file wsrf-engine.config:
a. Adding MSS Server URL.

```
<wsag4j-config:GatewayAddress>http://packcs-e0.scai.fraunhofer.de/wsag4unicore6-agreement-factory-
0.1.1/services</wsag4j-config:GatewayAddress>
```

b. Deletion of Extra empty tag `<wsag4j-config:GatewayAddress>`

c. Inserting both Keystore and Truststore files information in wsrf-engine.config
Note: Adding user certificates in keystore with UNICORE Client, the common utility "keytool" will not work.

3.2.2.2 *B: WISDOM Scheduler Server Deployment*

Installation:
Download WAR wisdom-scheduler-1.0.3.war from https://bpackcs-e0.scai.fraunhofer.de/wisdom-scheduler and deploy in a separate Tomcat instance. Deployment in single tomcat instance was not working due to an issue with the port already used by MSS Server.

Required Changes in Configuration files:
1. Editing the required configuration file wisdom.config:
a. Adding the MSS Server URL

```
  <wsag4u:Unicore6Configuration xmlns:wsag4u="http://schemas.scai.fraunhofer.de/wsag4unicore6/">
<wsag4u:Registry>
<wsag4u:Shared>false</wsag4u:Shared>
<wsag4u:URL>http://packcs-e0.scai.fraunhofer.de/wsag4unicore6-agreement-factory-0.1.1/services</wsag4u:URL>
<wsag4u:Description>WISDOM Registry</wsag4u:Description>
</wsag4u:Registry>
</wsag4u:Unicore6Configuration>
```

2. Editing required configuration file : wsrf-engine.config:
a. Adding the WISDOM-Scheduler gateway Address

```
<wsag4j-config:GatewayAddress>http://packcs-e0.scai.fraunhofer.de/wisdom-scheduler-1.0.3/services</wsag4j-
config:GatewayAddress>
```

b. Adding user keystore and Truststore files information in the file wsrf-engine.config. We use the same keystore and truststore files as used in wsrf-engine.config file of MSS Server.

Client side configuration:

To access the MSS Service for WISDOM applications from within UNICORE6 Rich Client, use the RCP plug-in/extension and then use the wisdom-scheduler URL as given below:
http://packcs-e0.scai.fraunhofer.de/wisdom-scheduler-1.0.3/services

3.2.3 DDSS

In the last stage of the PHOSPHORUS project a special DDSS Backup application has been developed. It integrated one of the applications used in the previous stages of the projects (DDSS GriddFTP using the G^{2}MPLS environment and visualizes the progress of the backup process (this integration is described in more detail in deliverable D3.9 - Final Report on the period M25 – M33)). The application was successfully shown on conferences and project reviews. Despite of the fact that the application runs only simple scenarios (files transfer – backup), the objectives have been achieved. This solution enables cooperation of resource reservation tool and backup application. The integration with the desktop management system Konqueror allows to perform backups in an easy way and to create own backup scenarios flexibly. Because of the fact that it is implemented in Python and Shell it can be seamlessly accessible from remote machines.

The application developed visualizes the process of link reservation and backup calls. A number of simple scenarios for the experiments has been prepared. Each scenario has been split into the following small steps:

- Initializing the DDSS backup synchronization utility: in this step data to backup are prepared, all transfer parameters are set and target backup server is chosen;
- Setting up a path to the chosen backup server: run signal over the possible path to destination server with request if it is possible to establish connection on all path length and then if response from all is yes, g2dialer is launched;
- Data plane connectivity check: when the path is reserved, it checks if it is possible to send data over the path (if data plane is ready to transmit data);
- Initialization of the GridFTP utility: create grid proxy and establish GridFTP connection;
- Making a backup on remote storage node: run GridFTP transfer;
- Closing the connection: tear down established path;
- Finalizing: it waits for control plane, till it clearly closes the connection and shows appropriate message.

3.3 Experiments conclusion

The PHOSPHORUS middleware UNICORE, MetaScheduling Service MSS and INCA have been fully functional since around month 28 of the project. The objectives of the WP3 internal middleware experiments during the last period of the project were (i) the consolidation of the modifications arising from the full integration with the HARMONY and G^{2}MPLS systems and (ii) improvements of non-functional properties like performance and stability. These objectives have been fully achieved, as proved by the demonstrations during the last phase.

As planned essential development of the applications and their integration in the PHOSPHORUS environment with the objective to allow applications to benefit from the advanced network technology of the PHOSPHORUS testbed - directly or through the middleware stack – was already finished around month 24 of the PHOSPHORUS project. During the last period of the project only minor changes were necessary to adapt the applications for full integration with the advanced network and middleware environment resulting from the integration of developments of WP1, WP2 and WP3. This leads to a number of impressing demonstrations during events like the Supercomputing Conference in Austin, November 17 to 21 2008 and the TERENA Network Conference in Malaga, June 8 to 11 2009.

4 Authentication, Authorisation and Accounting experiments – WP4

The following subchapters describe both the WP1 inter-domain data and control plane configuration used for the implemented WP1 Harmony AAA/AuthZ scenarios. This includes the topology information, naming spaces and addressing schemes. Afterwards the AAA/AuthZ scenarios themselves are depicted and the experiences received from the integration of the GAAA-TK library are given.

4.1 Inter-domain data plane configuration

4.1.1 Inter-domain connections: VLAN naming spaces

The different domains within the WP1 data plane are connected either directly, like the different VIOLA domains, or via Layer 2 Virtual Private Networks L2VPNs. Inter-domain 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	KISTI
6	UESSEX
7	VIOLA
8	CRC / HSVO
9	Internet2
10	UvA

Table 4.1: 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.

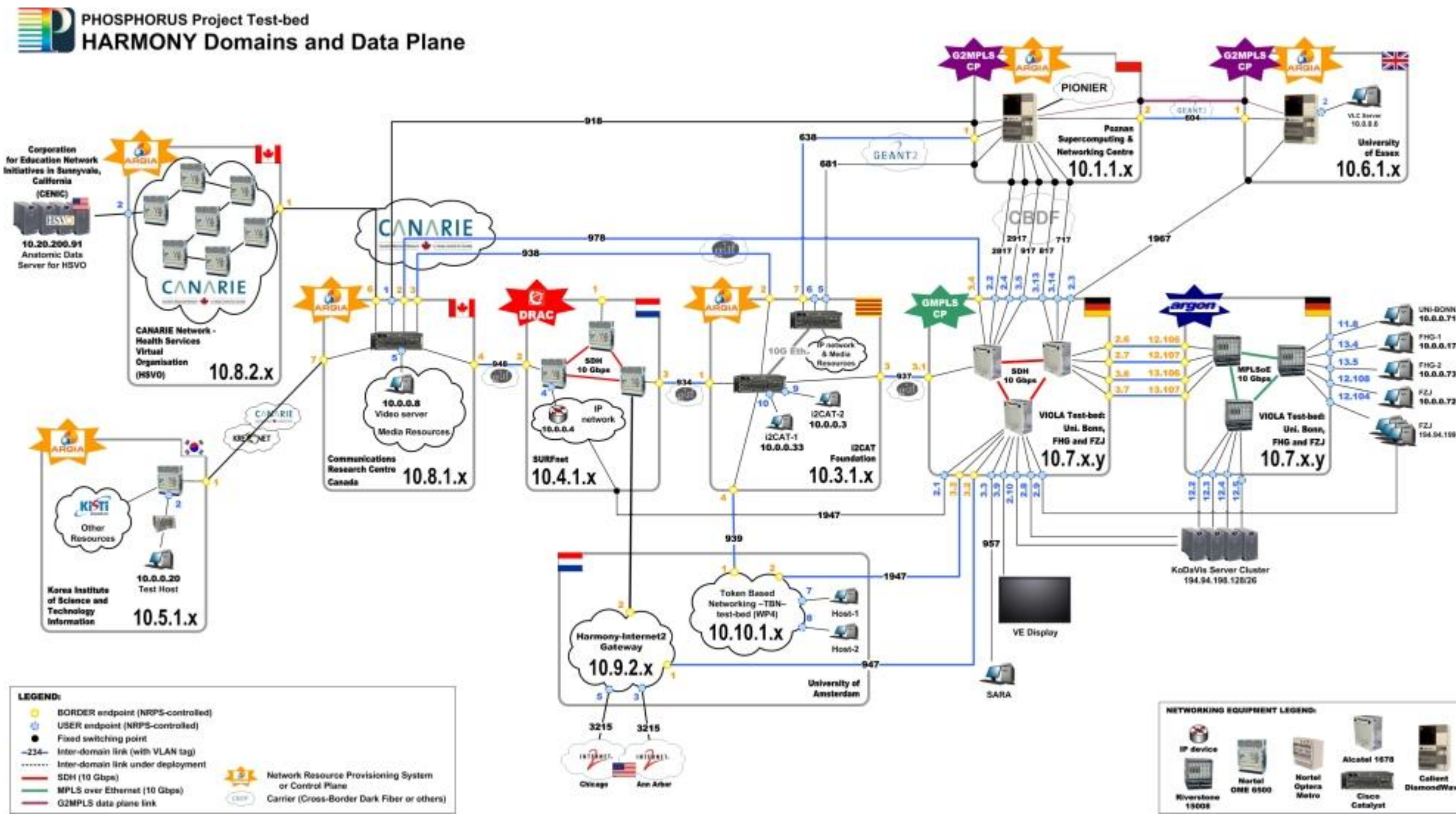
PHOSPHORUS Project Test-bed
HARMONY Domains and Data Plane
CANARIE
PIONIER
GEANT2
CBDF
DRAC
GMPLS CP
G2MPLS CP
ARGIA
argon
10.8.2.x
10.8.1.x
10.4.1.x
10.3.1.x
10.1.1.x
10.6.1.x
10.7.x.y
10.5.1.x
10.10.1.x
10.9.2.x
VIOLA Test-bed: Uni. Bonn, FHG and FZJ
University of Amsterdam
Harmony-Internet2 Gateway
Token Based Networking –TBN– test-bed (WP4)
KoDaVis Server Cluster 194.94.198.128/26
VE Display
SARA
Chicago
Ann Arbor
LEGEND:
BORDER endpoint (NRPS-controlled)
USER endpoint (NRPS-controlled)
Fixed switching point
Inter-domain link (with VLAN tag)
Inter-domain link under deployment
SDH (10 Gbps)
MPLS over Ethernet (10 Gbps)
G2MPLS data plane link
Network Resource Provisioning System or Control Plane
Carrier (Cross-Border Dark Fiber or others)
NETWORKING EQUIPMENT LEGEND:

Figure 4.1: Data plane overview

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 4.2: Original 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 (ID=7):	VLANs 717, 817, 917, 2817, 2917
• UESSEX (ID=6) to VIOLA (ID=7):	VLAN 1967
• SARA (ID=5) to VIOLA (ID=7):	VLAN 957
• UvA (ID=10) to I2CAT (ID=3):	VLAN 939

Additionally, the links to SURFnet (ID=4) from i2CAT and VIOLA have been reconfigured in order to integrate the Internet2 and UvA testbed:

• Internet2 (ID=9) to I2CAT (ID=3):	VLAN 934
• UvA (ID=10) to VIOLA (ID=7):	VLAN 1947
• Internet2 (ID=9) to VIOLA (ID=7)	VLAN 947

The connections between local testbeds used for NRPS tests are depicted in **Figure 4.2**.

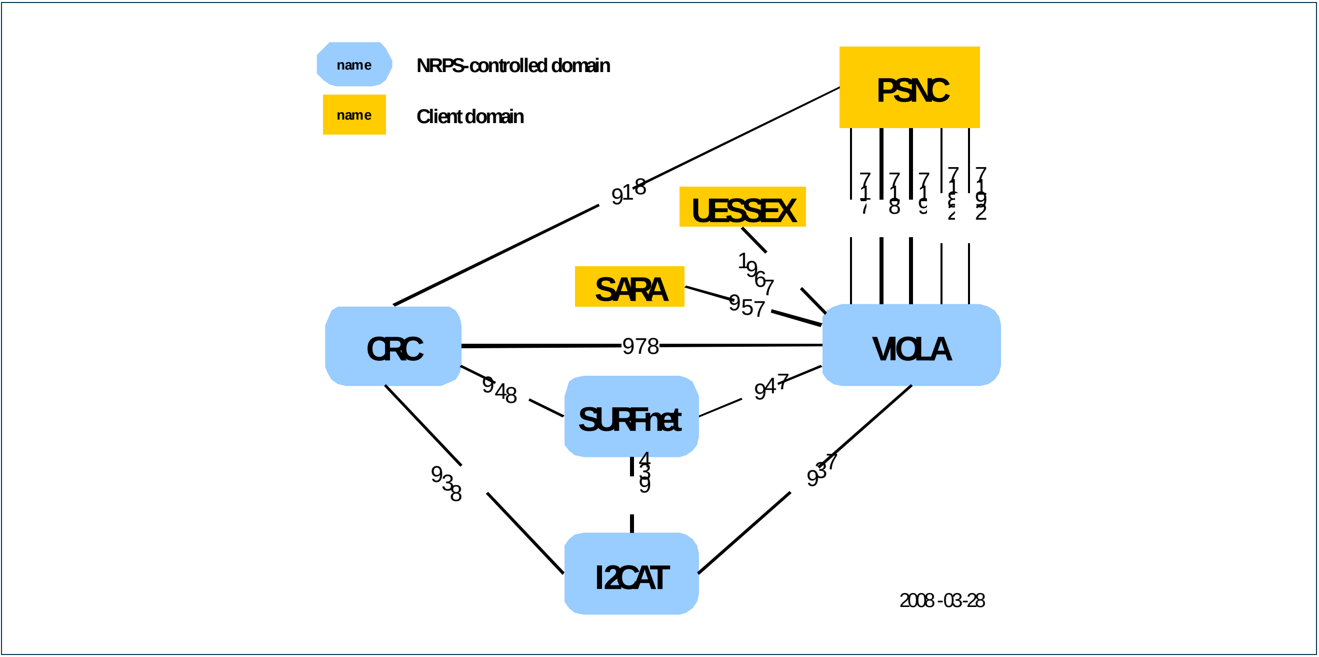

Figure 4.2: General VLAN map and addressing.

4.1.2 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	10.1.1.0 / 24	ARGIA / UCLP	10.1.1.0 / 24
3	I2CAT	10.3.1.0 / 24	ARGIA / UCLP	10.3.1.0 / 24
4	SURFnet	10.4.1.0 / 24	DRAC	10.4.1.0 / 24
5	KISTI	10.5.1.0 / 24	ARGIA / UCLP	10.5.1.0 / 24
6	UESSEX	10.6.1.0 / 24	ARGIA / UCLP	10.6.1.0 / 24
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	ARGIA / UCLP	10.8.1.0 / 24
	HSVO	10.8.2.0 / 24		10.8.2.0 / 24
9	Internet2	10.9.2.0/24	IDC/DC	10.9.2.0/24
10	UvA	10.10.1.0/24	*Does not apply*	

Table 4.3: 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, 10.0.0.173
• FZJ:	10.0.0.72
• Internet2:	10.0.0.91
• KISTI	10.0.0.20
• UESSEX	10.0.0.6

In **Figure 4.1** a full map of the PHOSPHORUS testbed can be found, in which VLAN tagging, TNA names and IP addressing are shown.

4.2 Inter-domain control plane configuration

Signalling between the domains participating in the WP1 testbed consists of web service calls between the NRPS Adapters and the central IDB instance. As described in detail in [Phosporus-D1.4], the NRPS Adapters register by calling the addOrEditDomain operations of the higher IDB instance's Topology Web Service (Topology-WS), and the central IDB 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, 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 4.4 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 (amy host)`	10.1.3.100
	`I2CAT (secured IDB)`	10.1.3.70
	`I2CAT (bender host)`	10.1.3.200
	`I2CAT (leela host)`	10.1.3.102
	`I2CAT (zoidberg host)`	10.1.3.201
Surfnet	`surfnet-testbed`	10.1.4.1
VIOLA	*(central IDB instance)*	10.1.7.1
	`viola-mpls`	10.1.7.2
	`viola-gmpls`	10.1.7.3
CRC	`CRC`	10.1.8.1
	`CRC (backup IDB)`	10.1.8.100
UESSEX	`uessex`	10.1.6.1

	`uessex (inca1)`	10.1.6.2
PSNC	`psnc`	10.1.1.100
KISTI	`kisti`	10.1.5.100

Table 4.4: Control plane addresses within the WP1 testbed

4.3 GAAA-TK Integration

As described in Sections 4.1 and 4.2 the WP1 testbed is composed of different domains following a specified numbering scheme and rudimentary security measures are already implemented. The latter and the more complex AuthN/AuthZ scenarios are described in the subsequent sections.

4.3.1 Level of Security

Security in this context can be located on the network, transport, and message level (cf. Figure 4.3). 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 4.2). Communication from other locations (e.g. for the user GUI) is allowed for a reduced set of source IP address ranges only. Other systems could easily be added to the VPN or to the allowed address range.

Since the security within the Service Plane is based on NLS, no **transport level security** (TLS) mechanisms are implemented. In order to communicate with other systems it is necessary to provide security for this level using SSL, but for the specific gateway only. This was exemplarily implemented within the scope of the Internet2 IDC Gateway/Translator.

In order to integrate the GAAA-TK into the WP1 Service Plane **message level security** (MLS) mechanisms were deployed. Since the GAAA-TK does not handle AuthN issues but is based on successful authentication, the following discussion is parted in the authentication (AuthN) and authorization (AuthZ) phase.

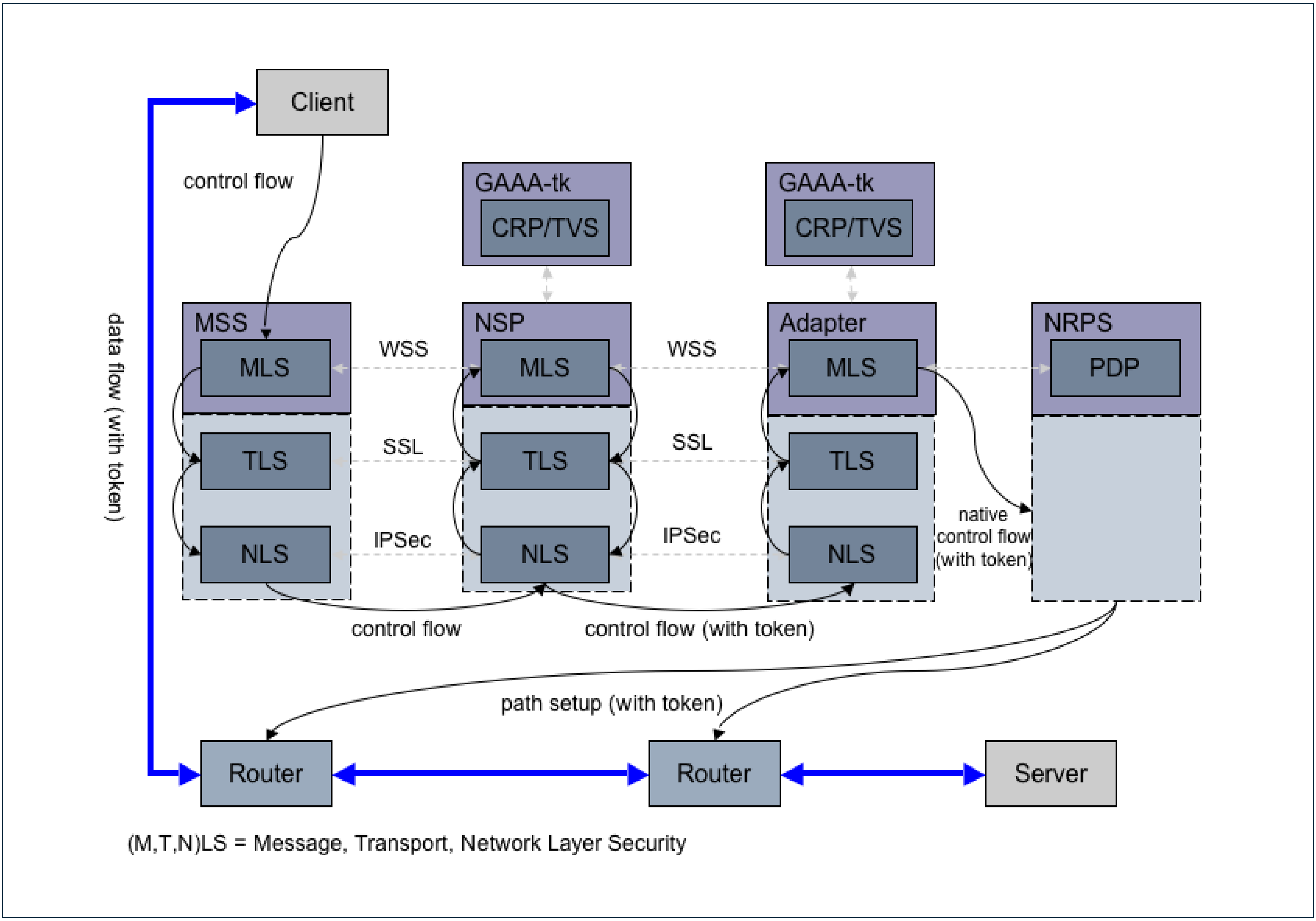

Figure 4.3: Overview of the security levels.

4.3.2 Authentication

The Harmony Service Interface contains 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 Simple Object Access Protocol (SOAP) messages the standard includes options to attach security credentials like username/password, X.509 certificates or tokens.

Figure 4.4 depicts a sequence of interactions needed for the authentication flow between interacting systems. The following sequence description is simplified and reduced to a single MSS (WP3 MetaScheduler), Harmony (WP1 Service Plane), and G^{2}MPLS (WP2 Service Plane) communication for AuthN aspects: (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 for the NSP/IDB and signs the message with its private key. (3) The message then is sent to the NSP/IDB on behalf of the client. Since the public key of the MSS is trusted within the IDB, the message is accepted in the next step. Furthermore a complex authorization process has to be implemented in order to validate the request. Finally 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/IDB. 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/IDB 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.

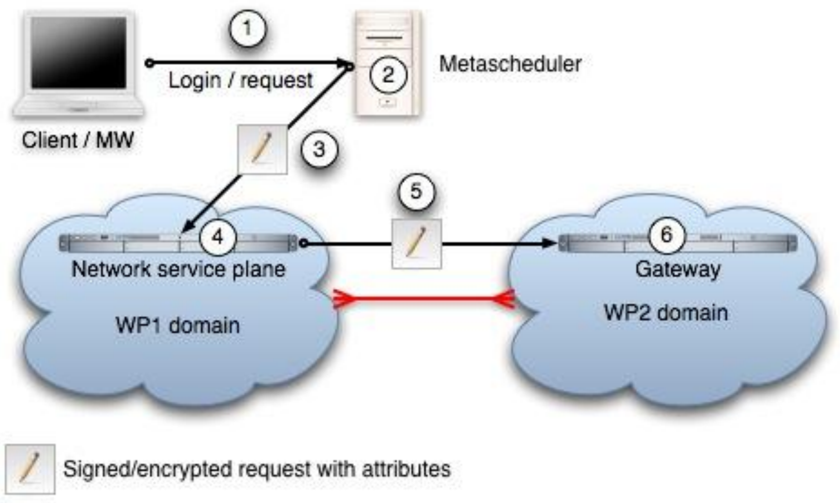

Figure 4.4: Authenticated message flow between MSS, NSP and G^{2}MPLS

4.3.3 Authorization

For the request **AuthZ** process WP1 is integrating the WP4s GAAA-TK into the Harmony Service Interface and will use it for all AuthZ related issues within the communication flow. As depicted in Figure 4.4 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/IDB forwards the request with the user credentials (attributes) that are contained in the incoming message from the middleware to the involved domains. It is assumed that each domain has its own policy and attribute database and they 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 the GAAA-TK, 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.

4.3.3.1 *Path creation*

In **Figure 4.5** 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/IDB/IDC/G^{2}MPLS-GWs public key. (2) The NSP/IDB/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 module by using the GAAA-TK. This message includes a global reservation identifier (GRI) created by the NSP/IDB/IDC/G^{2}MPLS-GW, the action, resources and credentials. The AuthZ server in return will send back a token. (4) Now the NSP/IDB/IDC/G^{2}MPLS-GW creates a new reservation request for the involved NRPSs/NSPs/IDBs/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/IDBs/G^{2}MPLS-GWs LRI, the GRI and the token for the reservation.

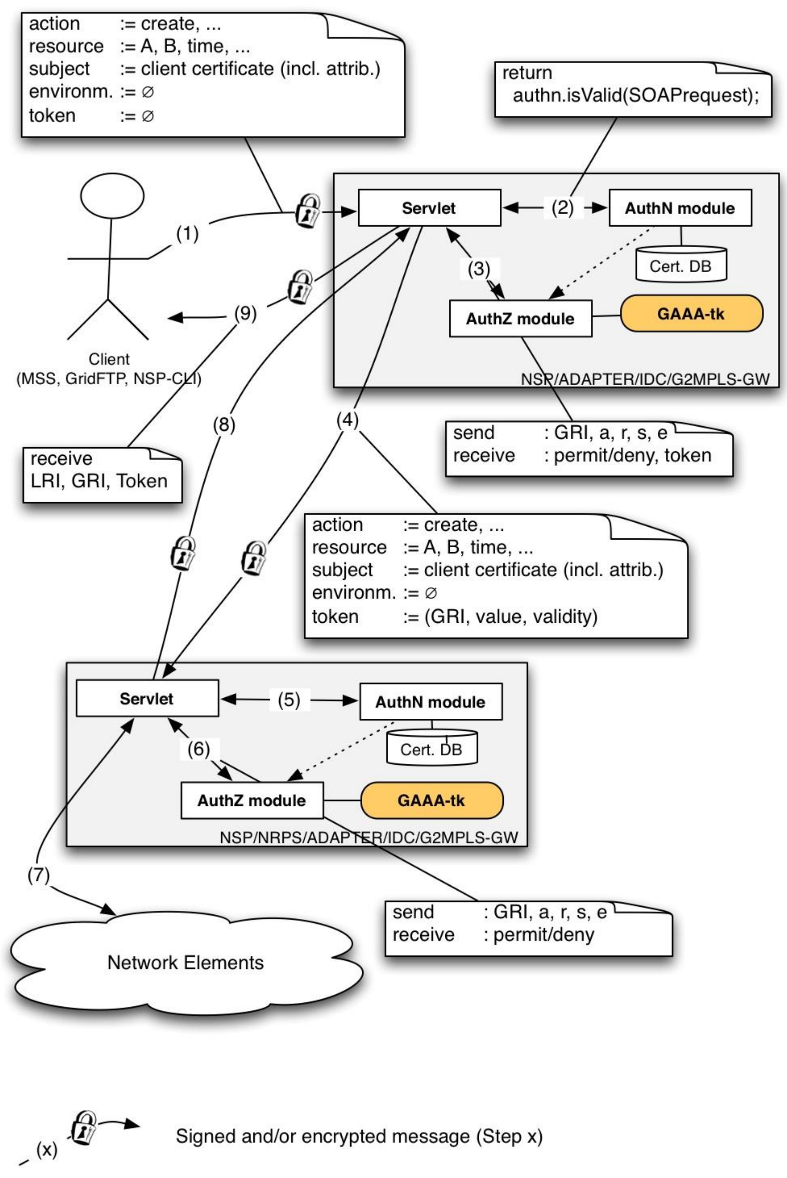
action := create, ...
resource := A, B, time, ...
subject := client certificate (incl. attrib.)
environm. := ∅
token := ∅
return authn.isValid(SOAPrequest);
Servlet
(2)
AuthN module
Cert. DB
(3)
AuthZ module
GAAA-tk
NSP/ADAPTER/IDC/G2MPLS-GW
(1)
(9)
Client (MSS, GridFTP, NSP-CLI)
receive LRI, GRI, Token
(8)
(4)
send : GRI, a, r, s, e
receive : permit/deny, token
action := create, ...
resource := A, B, time, ...
subject := client certificate (incl. attrib.)
environm. := ∅
token := (GRI, value, validity)
Servlet
(5)
AuthN module
Cert. DB
(6)
AuthZ module
GAAA-tk
NSP/NRPS/ADAPTER/IDC/G2MPLS-GW
(7)
send : GRI, a, r, s, e
receive : permit/deny
Network Elements
(x)
Signed and/or encrypted message (Step x)

Figure 4.5: Generalized AuthN/AuthZ workflow for path creation

4.3.3.2 *Path administration*

In **Figure 4.6** 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/IDBs/IDCs/G^{2}MPLS-GWs public key. (2) The NSP/

IDBs/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/IDBs/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/IDBs/IDC/G^{2}MPLS-GW in step (8). (9) Finally the client receives the confirmation for the activation.

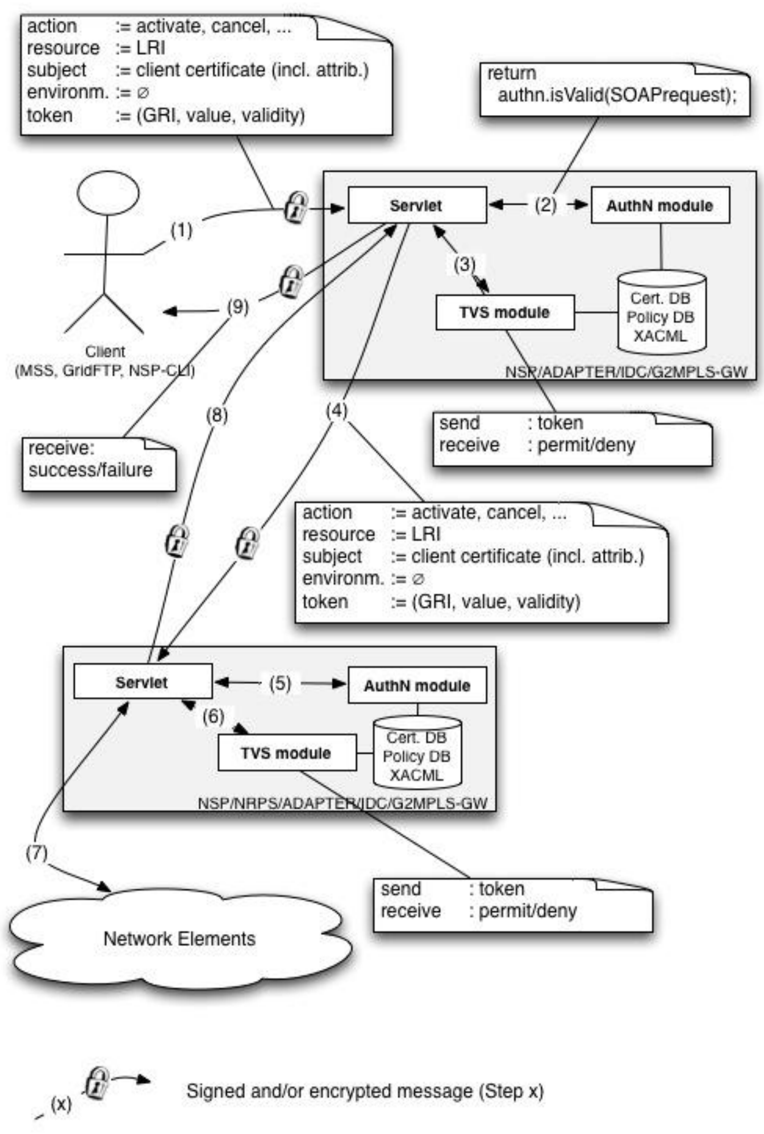
action := activate, cancel, ...
resource := LRI
subject := client certificate (incl. attrib.)
environm. := ∅
token := (GRI, value, validity)
return authn.isValid(SOAPrequest);
Servlet
AuthN module
TVS module
Cert. DB
Policy DB
XACML
NSP/ADAPTER/IDC/G2MPLS-GW
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)
Client
(MSS, GridFTP, NSP-CLI)
receive: success/failure
send : token
receive : permit/deny
action := activate, cancel, ...
resource := LRI
subject := client certificate (incl. attrib.)
environm. := ∅
token := (GRI, value, validity)
Servlet
AuthN module
TVS module
Cert. DB
Policy DB
XACML
NSP/NRPS/ADAPTER/IDC/G2MPLS-GW
send : token
receive : permit/deny
Network Elements
(x)
Signed and/or encrypted message (Step x)

Figure 4.6: Generalized AuthN/AuthZ workflow for path administration

4.3.3.3 *Multi-domain GAAA-TK Integration*

In **Figure 4.7** and **Figure 4.8** the current path creation and path administration scenarios for WP1 are depicted. They show how messages are forwarded between different domains and how the GAAA-TK is integrated in the different AuthZ stages. It's important to note that the GRI, in contrast to what is described in D4.2 Section 2.1, is not generated at the beginning. Instead it is generated after the underlying system has confirmed the reservation. This is why the AuthZ process is divided into two steps (3.5) and (9.5) in **Figure 4.7**.

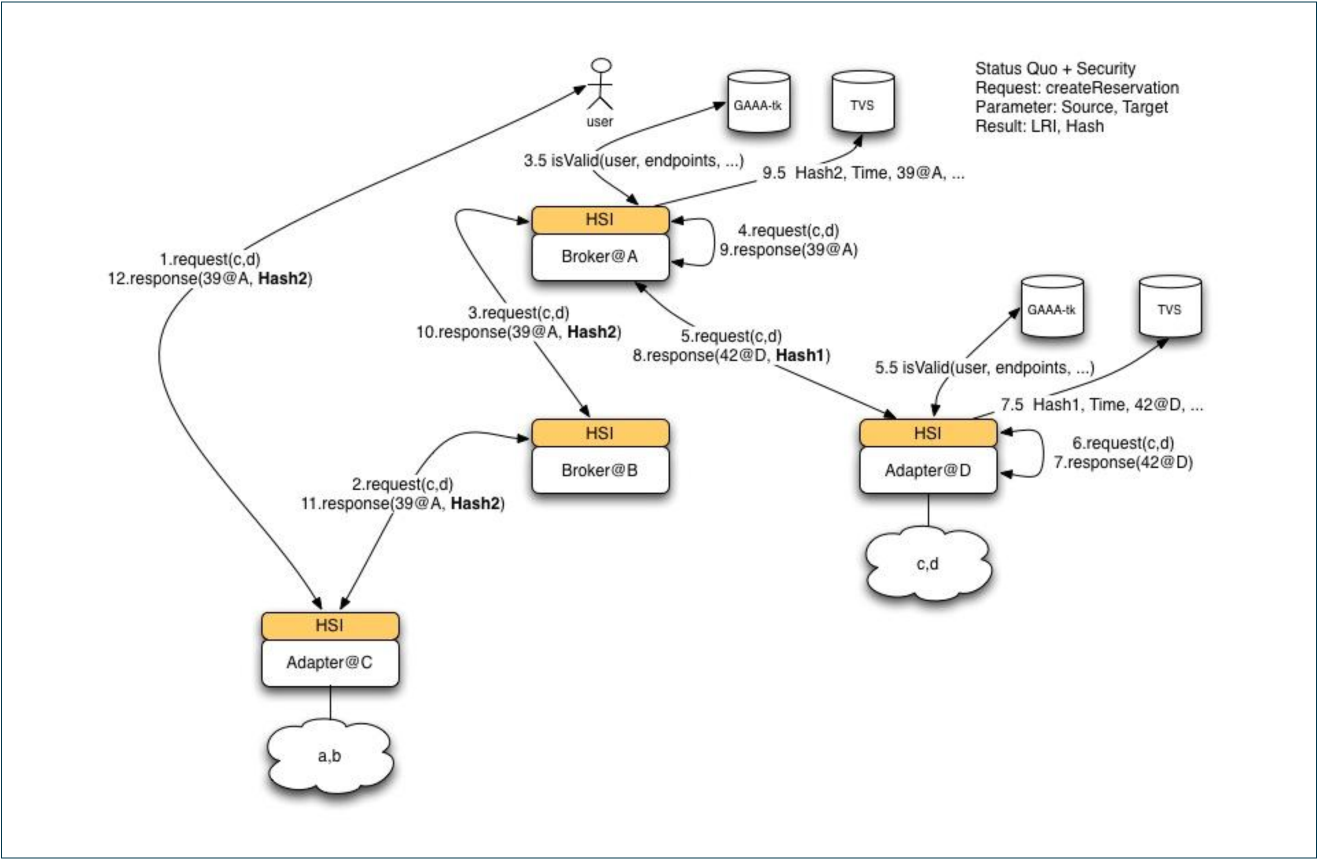

Figure 4.7: Multi-domain GAAA-TK Integration Scenario (createReservation)

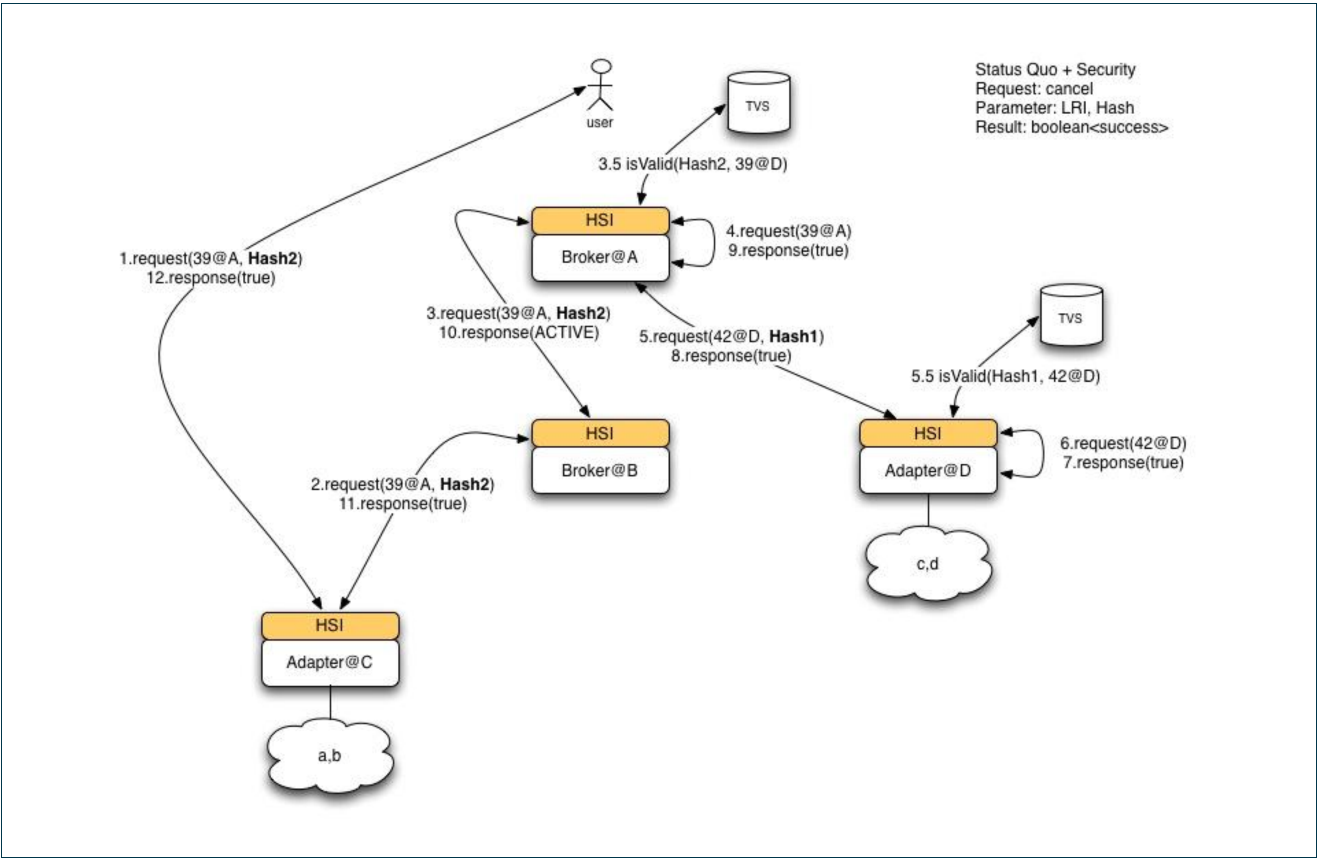

Figure 4.8: Multi-Domain GAAA-TK Integration Scenario (cancelReservation)

4.4 Token Based Networking (TBN) integration

4.4.1 The Token Based Networking architecture

The Token Based Networking (TBN) architecture was initially introduced to establish lightpaths over multiple network domains [CRISTEA-2007]. Lightpaths are setup on behalf of authorised applications that need to bypass transit networks. On the one hand, TBN uses a secure signature of pieces of an IP packet as a token that is placed inside the packet to recognise and authenticate traffic. The applications traffic is first tokenised by the TokenBuilder of a local domain (e.g., a campus network), after which it is enforced by the TokenSwitch at each inter-domain controller along the end-to-end path. On the other hand, TBN makes use of a separate service and control plane. The control plane consists of an AAA server in the push sequence as explained by the Authorisation Authentication Accounting (AAA) framework (RFC 2904). The AAA server acts as an authority that is responsible for the reservation and provisioning of the end-to-end paths, possibly spanning multiple network domains.

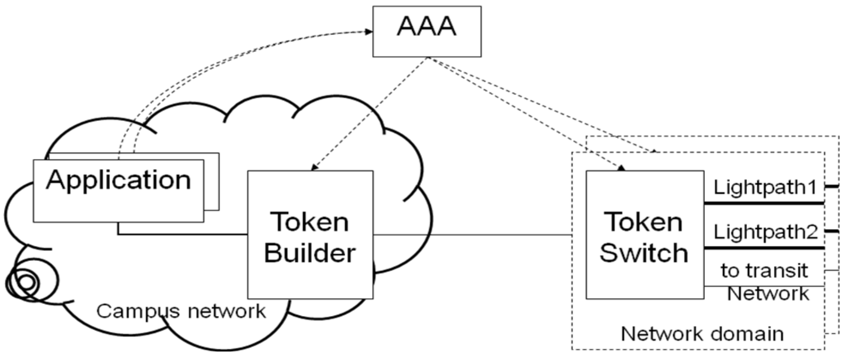

Figure 4.9: The Token Based Networking architecture

Figure 4.9 shows that, on behalf of an application, the AAA authority provisions the network resources required for an end-to-end lightpath that may cross multiple domains. At runtime, the application traffic is first tokenised by TokenBuilder and then subsequently enforced by each TokenSwitch along the lightpath.

Making tokens protocol independent has an important advantage; the token can be regarded as an aggregation identifier to a network service. Generally, we see four types of aggregation identifiers that can be combined, as follows:

- identifier to point a service to the Network Element – NE (e.g., a multi-cast, or transcoding);
- identifier that defines the service consumer (e.g., the GRID application);
- identifier that defines the serviced object (e.g., the network stream);
- identifier that defines the QoS (security, authorisation, robustness, deterministic property, etc.).

First, a token can bind to different semantics and services (e.g., network services for a user, a group of users, or an institute). The semantics that is referred to by a token (e.g., a certain routing behaviour) can be hard-coded into a switch or the token can refer to a stored aggregation identifier that points at the specific behaviour in a programmable network device. Hence, a token provides a generic way to match applications to their associated network services. Second, tokens can be either embedded in the application generated traffic or encapsulated in protocols where embedding is not supported, such as in public networks. Third, tokens can also provide a general type of trusted and cryptographically protected proof of authorisation with flexible usage policies.

4.4.2 Token Based Network (TBN) testbed

The Token Based Network (TBN) testbed uses Token Based Switch over IP (TBS-IP) as a low-level system for traffic switching at high speeds (multi gigabits/sec) based on packet authentication. TBS-IP helps high-performance computing and GRID applications that require high bandwidth links between GRID nodes to use priority links for authorised packets with policy constraints. TBS-IP is fast and safe and uses the latest network processor generation (Intel IXP2850). Figure 1 shows an overview of how the TBN testbed is plugged into the PHOSPHORUS GMPLS testbed. The TBN testbed works as a separate network domain (UvA-TBN) which interconnects an entire UvA IP campus network into the GMPLS testbed. In other words, applications running on various hosts connected within UvA IP campus network may request, and authorise to use GMPLS lightpaths towards other end-points into the European PHOSPHORUS testbed, such as I2CAT (Barcelona), or VIOLA (Bonn).

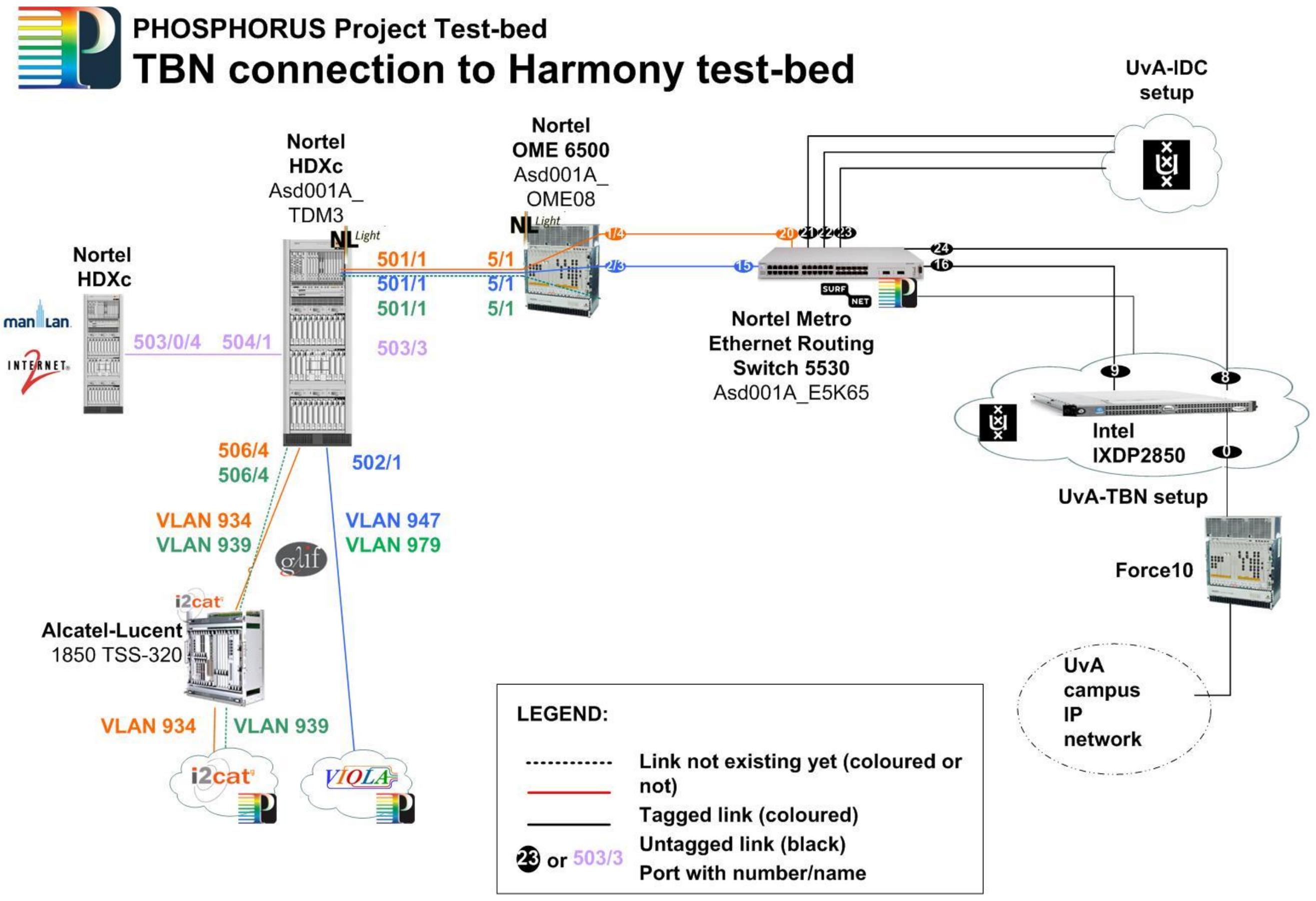

Figure 4.10: TBN integration in Harmony testbed

Note that in this testbed, TBN works as a gateway router between IP networks and GMPLS circuits, and therefore, the mapping of IP over GMPLS uses specific AAA authorisation sequences implemented by a high-level authority, Harmony, at service plane. An example of authorization sequence is illustrated in Figure 4.11 and described below.

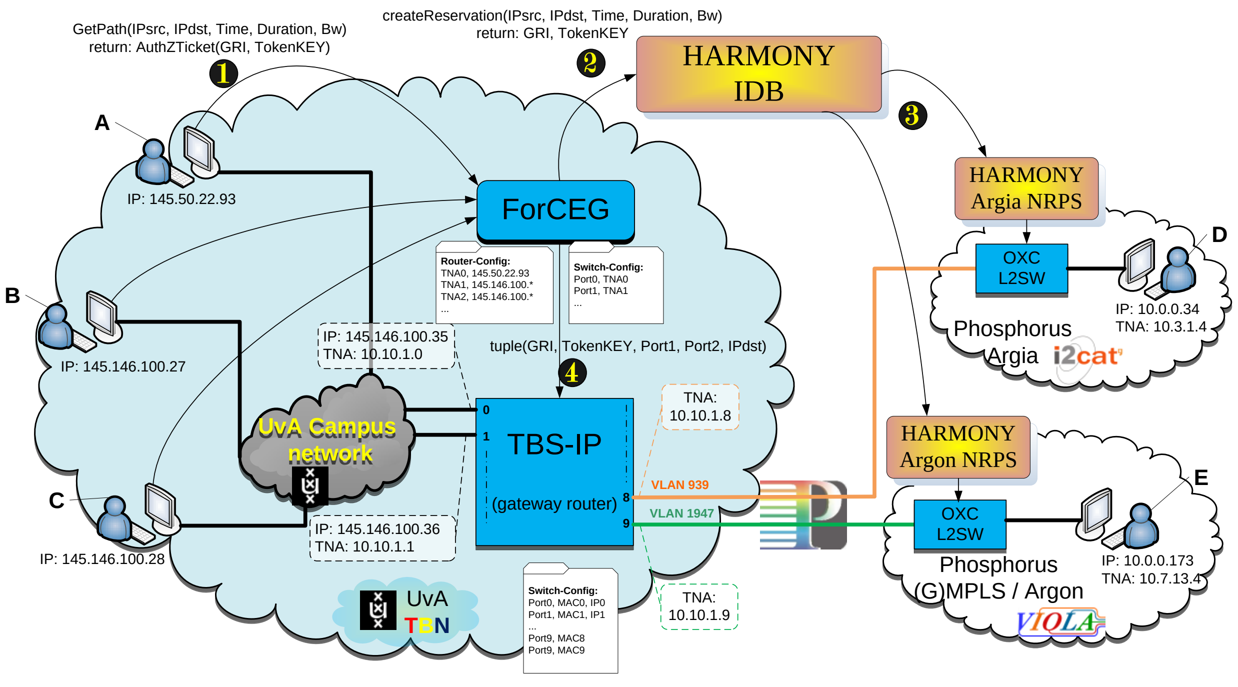
GetPath(IPsrc, IPdst, Time, Duration, Bw)
return: AuthZTicket(GRI, TokenKEY)
createReservation(IPsrc, IPdst, Time, Duration, Bw)
return: GRI, TokenKEY
HARMONY IDB
ForCEG
HARMONY Argia NRPS
HARMONY Argon NRPS
OXC L2SW
Phosphorus Argia
Phosphorus (G)MPLS / Argon
A
B
C
D
E
IP: 145.50.22.93
IP: 145.146.100.27
IP: 145.146.100.28
IP: 145.146.100.35 TNA: 10.10.1.0
IP: 145.146.100.36 TNA: 10.10.1.1
IP: 10.0.0.34 TNA: 10.3.1.4
IP: 10.0.0.173 TNA: 10.7.13.4
tuple(GRI, TokenKEY, Port1, Port2, IPdst)
TBS-IP
(gateway router)
UvA Campus network
UvA TBN
TNA: 10.10.1.8
TNA: 10.10.1.9
VLAN 939
VLAN 1947
Router-Config: TNA0, 145.50.22.93 TNA1, 145.146.100.* TNA2, 145.146.100.*
Switch-Config: Port0, TNA0 Port1, TNA1
Switch-Config: Port0, MAC0, IP0 Port1, MAC1, IP1 Port9, MAC8 Port9, MAC9

Figure 4.11: Testbed configuration for testing a TBN domain with multiple applications sharing 2 lightpaths provisioned through PHOSPHORUS testbed

Some specific user applications (e.g., VideoLAN server) from IP campus network want to connect to VideoLAN Clients (VLC) located in other network domains interconnected over GMPLS as in Figure 4.11. The work-flow of authorisation of the user-applications to use the network services uses the following four steps:

1. User application requests to the UvA-TBN gateway router (ForCEG service in Figure2) a path between the IP addresses (IPsrc, IPdst, startTime, duration, bandwidth), where IPsrc must be within its IP campus network, and IPdst should be available within the GMPLS networks.

2. The ForCEG service will authenticate the user application and request a path to the GMPLS authority (IDB in Figure2) on behalf of its user applications. As a result, IDB will generate and return the credentials of using authorised paths in format of a Global Reservation Identifier (GRI) and a TokenKey used of in-band encryption.

Notice that ForCEG keeps a local cache of the mappings IPs to TNAs related to the TBN domain and hence, it is able to send a request for GMPLS circuits in format of (TNA1, TNA2, startTime, duration, bw). However, delegating the mapping table up to the GMPLS authority (IDB) would assure the correctness and consistency of IPs to TNAs mapping across all the domains within the entire GMPLS networks.

3. IDB will prepare all intermediate domains involved in the path provisioning with the requested credentials (GRI, TokenKey, etc).

4. Once the start-time of the requested path arrives, the received credentials from IDB are enforced into the TBS-IP gateway router at the lowest level (IP packets and circuits).

5. When a user application (e.g., vlc) is authorised to connect to an IPdst over GMPLS network, the magic-carpet environment inserts encrypted tokens within the outgoing packets (media streams) and redirect them towards TBS-IP.

6. TBS-IP authenticates every received packet identified by the plain-text GRI as a lookup computation (finds GRI-entry into the local AuthZ-table), then it does:

    a. If the Packet comes from IPcampus network (PKT.IPdst==GRI-entry.Port1.IPaddr), then it will re-write its IPdst to the correct endpoint (PKT.IPdst=GRI-entry.IPdst) and will push the packet into the proper GMPLS path, as indicated by GRI-entry.Port2;

    b. If the Packet comes from GMPLS network (PKT.IPdst!=GRI-entry.Port2.IPaddr), then it will push the packet into the proper outgoing port into IPcampus as indicated by GRI-entry.Port1;

4.5 Use Cases and Test-bed Experiences

The current naming and addressing scheme of the WP1 Harmony testbed were successful transformed into a TNA based XACML-NRP policy profile. It permits reservations for a selected set of TNA address ranges (= corresponding domain) and actions for PHOSPHORUS testbed users with specific roles. Thereby the following two scenarios are supported:

- Use Case 1: User/Group A is only allowed to use endpoints X, Y and Z

- Use Case 2: User/Group A is only allowed to use endpoints in domain N and M

Furthermore, after some iterations of the library, it was successful integrated into the Harmony Service Interface (HSI) and consequently part of each Harmony IDB, Adapter, and Translator. The information that are needed by the GAAA-TK are extracted from the incoming request signature by the AuthN module (subjectId and subjectRole), the Harmony Request (action, endpoints, validityTime), and the Harmony Response (GRI). After filling the TVS table with the required information, the generated Token is sent back to the user within the response message. The subsequent cancel request was authorized by matching the TVS table information with the given action, subjectId, subjectRole, GRI, Token and validityTime.

Unfortunately with the current version of the GAAA-TK TVS each Token is only valid for a single action (e.g. cancel). This can be regarded as a requirement to the AuthZ decision integrity - according to this the authorisation and ticket is granted to what was requested. Extending this decision can be treated as a kind of delegation and typically entails either using special delegation policies or policy obligations. This requirement is crucial to use the GAAA-TK TVS within the WP1 testbed, since several actions are needed to administrate existing reservations. This leads us to other scenarios:

- Use Case 3: User/Group A is only allowed to invoke method X, Y, and Z
- Use Case 4: User/Group A is only allowed to invoke method X,Y, and Z based on session delegation

Finally all possible requests that are used within the Harmony interface (abbreviated in Use Case 3+4 as X, Y, and Z) must have a corresponding mapping within the XACML NRP policy description of each domain. After that and after solving minor technical issues, nothing should be in the way to fully integrate the GAAA-TK into the main WP1 testbed.

5 Integration between Control Plane and Network Resource Provisioning Systems

This section provides a description of the tests carried out to demonstrate the feasibility of interoperation of the Harmony system implemented in WP1 with G^2MPLS implemented in WP2. The tests have been performed setting an overlay scenario in which G^2MPLS operates as a domain under the control of Harmony.

5.1 Testing environment and test-bed setup

G^2MPLS Control Plane prototype implemented in WP2 includes all the designed functionalities. It consists of all the software modules designed, implemented and publicly demonstrated, released in the form of a software package. Four different kind of controllers can be run depending just on the node configuration: G^2MPLS UNI-C controller, G^2MPLS edge controller, G^2MPLS core controller, and G^2MPLS border controller.

On the other hand, WP1 has implemented Harmony service architecture with the set of interfaces for interoperability in a seamless environment between different NRPS and the standard GMPLS towards the GRID Middleware.

The integration has been achieved by introducing the Harmony-G^2MPLS Gateway (HG^2GW) that implements the Harmony reservation web service and the E-NNI G^2MPLS CORBA client/servant to enable the interoperation of both systems.

In order to demonstrate the feasibility of interconnecting heterogeneous domains (e.g. ARGIA, G^2MPLS) controlled under the umbrella of Harmony the test-bed setup includes the Health Services Virtual Organization (HSVO) application which consists in a remote server and visualization client. The client is deployed at Fi2CAT while the remote server is located in Sunnyvale, California with a direct connection to CRC in Canada. Thus, the computed path crosses multiple domains (CRC – Canada, PSNC – Poland, i2CAT – Barcelona) controlled by its own provisioning system or control plane. Specifically, CRC and i2CAT domains use ARGIA (Harmony) and PSNC uses G^2MPLS to control its Ethernet domain.

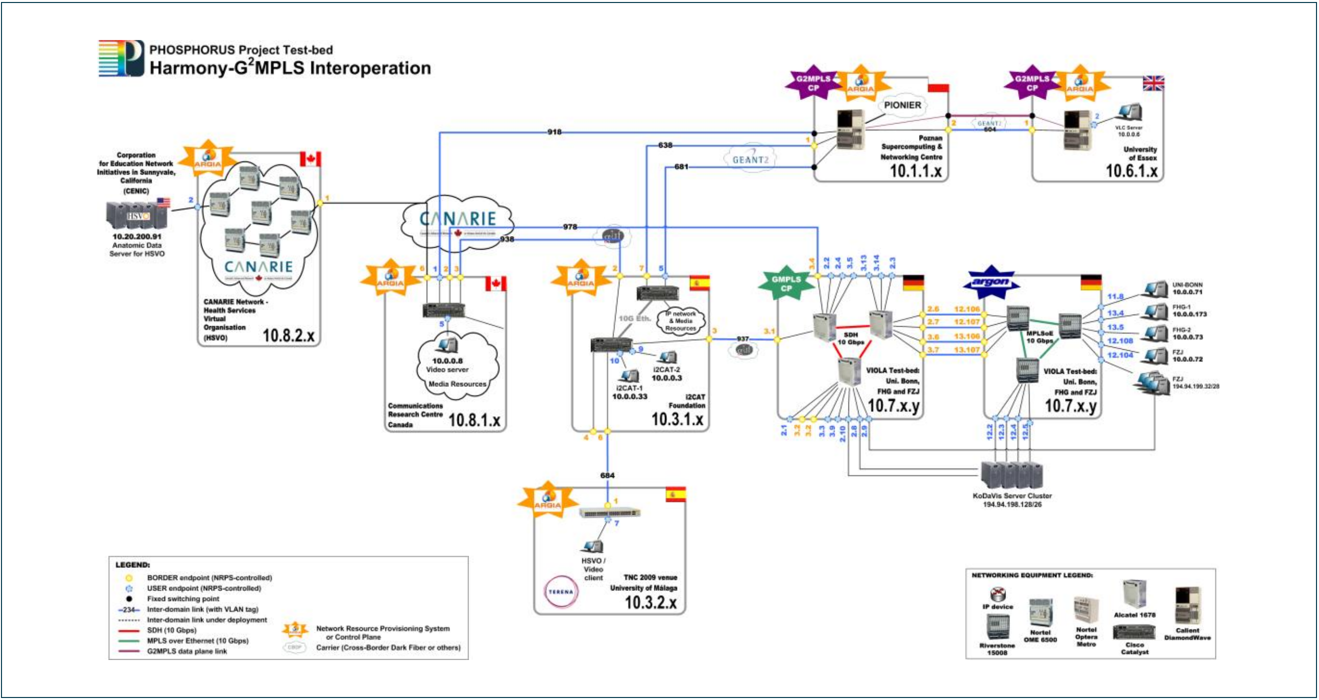
PHOSPHORUS Project Test-bed
Harmony-G2MPLS Interoperation
Corporation for Education Network Initiatives in Sunnyvale, California (CENIC)
10.20.200.91 Anatomic Data Server for HSVO
CANARIE Network - Health Services Virtual Organisation (HSVO)
10.8.2.x
CANARIE
Communications Research Centre Canada
10.8.1.x
10.0.0.8 Video server
Media Resources
i2CAT Foundation
10.3.1.x
i2CAT-1 10.0.0.33
i2CAT-2 10.0.0.3
10G Eth.
IP network & Media Resources
HSVO / Video client
TNC 2009 venue University of Málaga
10.3.2.x
PIONIER
Poznan Supercomputing & Networking Centre
10.1.1.x
GEANT2
University of Essex
10.6.1.x
VLC Server 10.0.0.6
G2MPLS CP
GMPLS CP
argon
ARGIA
VIOLA Test-bed: Uni. Bonn, FHG and FZJ
10.7.x.y
SDH 10 Gbps
MPLSoE 10 Gbps
UNI-BONN 10.0.0.71
FHG-1 10.0.0.173
FHG-2 10.0.0.73
FZJ 10.0.0.72
FZJ 194.94.199.32/28
KoDaVis Server Cluster 194.94.198.128/26
LEGEND:
BORDER endpoint (NRPS-controlled)
USER endpoint (NRPS-controlled)
Fixed switching point
Inter-domain link (with VLAN tag)
Inter-domain link under deployment
SDH (10 Gbps)
MPLS over Ethernet (10 Gbps)
G2MPLS data plane link
Network Resource Provisioning System or Control Plane
Carrier (Cross-Border Dark Fiber or others)
NETWORKING EQUIPMENT LEGEND:
IP device
Riverstone 15008
Nortel OME 6500
Nortel Optera Metro
Alcatel 1678
Cisco Catalyst
Calient DiamondWave
918
638
681
978
938
937
684
604

Figure 5.1: Test-bed topology

5.2 Functional tests

The performed tests verify the functionalities introduced by HG^{2}GW to prove the correct interoperation between Harmony and G^{2}MPLS systems. This chapter just focuses in the relevant inter-domain operations related with Harmony-G^{2}MPLS integration which include Reservation Creation, Reservation Cancellation and Get Connection Status. The actual G^{2}MPLS operation has already been described and tested previously and thus is out of the scope of the description of these tests.

5.2.1 Reservation Creation

When a connection request managed by Harmony has to cross a G^{2}MPLS domain previously registered in the Harmony IDB, the HG^{2}GW Reservation Web Service is used to set up the segment path within G^{2}MPLS. The Reservation Web Service is the same used by Harmony to set up the other NRPS segments, so HG^{2}GW acts as a proxy to adapt the request to G^{2}MPLS CORBA calls that signal the call and LSP establishment. The required parameters are source and destination TNAs, Bandwidth and scheduling calendar if necessary. The following log shows the correct transmission and CORBA method invocation performed by HG^{2}GW:

```
2009/05/08 12:23:40 HG2GW: [DBG] nRes Call: createReservation
2009/05/08 12:23:40 HG2GW: [DBG] ServiceID = 1
2009/05/08 12:23:40 HG2GW: [DBG] Bandwidth = 800
```

```
2009/05/08 12:23:40 HG2GW: [DBG] Going to create a HG2GW CALL with localID 1
2009/05/08 12:23:40 HG2GW: [DBG] Sent callCreate through CORBA Mgmt interface
2009/05/08 12:23:40 HG2GW: [DBG] Created call with identity [OPERATOR SPECIFIC]
2009/05/08 12:23:40 HG2GW: [DBG]         - srcAddr             : (IPv4) 192.168.101.201/32
2009/05/08 12:23:40 HG2GW: [DBG]         - localId             : 0x0000000000000001
2009/05/08 12:23:40 HG2GW: [DBG] Sent callSetTna for iTNA through CORBA Mgmt interface
2009/05/08 12:23:40 HG2GW: [DBG] Sent callSetTna for eTNA through CORBA Mgmt interface
2009/05/08 12:23:40 HG2GW: [DBG] Sent callSetUp through CORBA Mgmt interface
2009/05/08  12:23:43  HG2GW:  [DBG]  Received  callRaiseMsgEvent(SetupIndication)  through  CORBA  EW
interface
2009/05/08 12:23:43 HG2GW: [DBG] Sent callRaiseMsgEvent(SetupConfirm) through CORBA EW interface
2009/05/08 12:23:43 HG2GW: [DBG] nRes Call: Returning createReservation
```

5.2.2 Reservation Cancellation

The reservation cancellation is performed in a similar way as the reservation creation. This time, just the Reservation ID is needed:

```
2009/05/08 12:26:23 HG2GW: [DBG] nRes Call: cancelReservation
2009/05/08 12:26:23 HG2GW: [DBG] Going to destroy a HG2GW CALL with localID 1
2009/05/08 12:26:23 HG2GW: [DBG] Sent callDestroy through CORBA Mgmt interface
2009/05/08 12:26:23 HG2GW: [DBG] nRes Call: Returning cancelReservation
```

5.2.3 Get Connection Status

With the get connection status operation, Harmony is able to request via HG^{2}GW the status of a particular reservation previously sent to G^{2}MPLS:

```
2009/05/08 12:25:11 HG2GW: [DBG] nRes Call: getStatus
2009/05/08 12:25:11 HG2GW: [DBG] Sent callGetTna through CORBA Mgmt interface
2009/05/08 12:25:11 HG2GW: [DBG] operstate: UP
2009/05/08 12:25:11 HG2GW: [DBG] nRes Call: Returning getStatus
```

5.2.4 Reservation request failure

When a request cannot be processed by G^{2}MPLS due to lack of resources, because the Job ID already exists or a generic error, Harmony is informed with an exception:

```
2009/05/08 12:29:23 HG2GW: [DBG] nRes Call: createReservation
2009/05/08 12:29:23 HG2GW: [DBG] ServiceID = 1
2009/05/08 12:29:23 HG2GW: [DBG] Bandwidth = 100
```

```
2009/05/08 12:29:23 HG2GW: [DBG] Going to create a HG2GW CALL with localID 1
2009/05/08 12:29:23 HG2GW: [DBG] Sent callCreate through CORBA Mgmt interface
2009/05/08 12:29:23 HG2GW: [ERR] callCreate through CORBA Mgmt interface failed for fetching problems
(reason= call already exists at me)
2009/05/08 12:29:23 HG2GW: [DBG] Sending exception...
2009/05/08 12:29:23 HG2GW: [DBG] nRes Call: Returning createReservation
```

5.3 Test Conclusions

All the performed tests have successfully validated the expected functionalities of HG^{2}GW to enable the interoperation of Harmony and G^{2}MPLS domains. Some issues regarding compatibility of the developing tools used at both sides have been solved with extra adaptation code. These activities have demonstrated the feasibility of establishing multi-domain end-to-end connections in a heterogeneous environment with different provisioning systems involved.

6 Integration between Network Resource Provisioning Systems and middleware and applications

The WISDOM application was selected as the use case for demonstrating the benefits of the integration between the GRID middleware and the Network Resource Provisioning Systems (NRPS). However, the framework is transparent for the application. The applications themselves don't have to be modified. To benefit from the PHOSPHORUS framework a user only needs to modify the client, which is used to submit jobs to allow for comfortably specifying the workflow and the parameters of an application. In case of a UNICORE environment the user simply creates a new application specific plug-in for the eclipse-based rich client.

The integration between Network Resource Provisioning Systems and GRIDs and applications allows defining High Throughput Virtual Screening workflows, which benefit both from the distribution of computational load required for the docking simulation and from the manageable QoS of the network through the MetaScheduling Service. Thus, the performance of workflow can be improved on the level of computational resources through reserving the most appropriate ones for the execution of the different docking simulations but also by reserving the network bandwidth between the source of the input data and the nodes where the data has to be transferred prior to the execution of the simulation. Moreover, the huge amount of output data resulting from the simulations may equally efficient transported back to the user's environment through reserved network connections with the required bandwidth, which become available at the end of the simulations.

The scenario is as follows:

- The user specifies his WISDOM job using the UNICORE client
- From the description the MetaScheduling Service creates a workflow consisting of
    - The transfer of the input data from the user's environment to the execution sites (stage-in)
    - The submission of the WISDOM jobs to the execution site
    - The transfer of the output data from the execution sites to the user's environment

The resulting workflow is depicted in **Figure 6.1**.

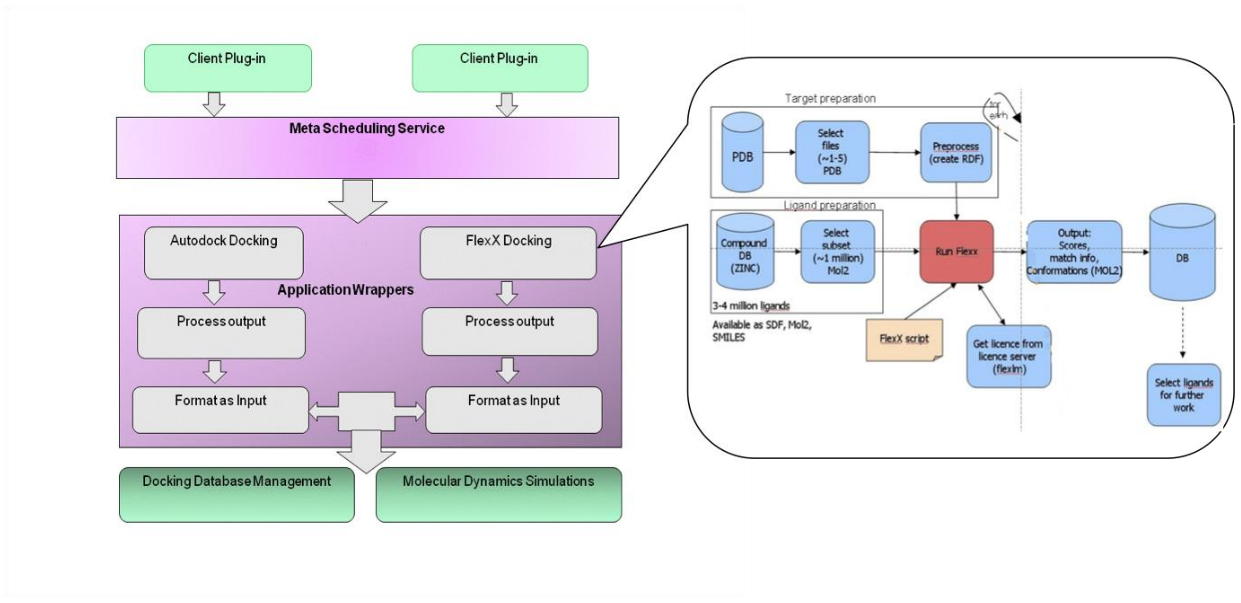

Figure 6.1: High throughput Virtual Screening workflow

The key actors for the integration between the GRID middleware and the NRPS are the MetaScheduling Service and HARMONY.

The MetaScheduling Service is responsible for decomposing the user's job description done in the GRID middleware client. Once the individual components of the workflow have been identified the MetaScheduling Service starts negotiating with the individual GRID resource management systems and the HARMONY system the availability of the different resources:

- For the computational resources the number of nods and their availability in time

- For the network the links between the node where the input data is located and where the output data will be consolidated and the individual nodes where the docking simulation is executed and the available bandwidth in time

The negotiation between the MetaScheduling Service and the different resource management systems and the HARMONY system (or their respective adapters) is using WS-Agreement as protocol. The successful negotiations result in individual Service Level Agreements between the MetaScheduling Service and the different resource management systems and the HARMONY system.

These Service Level Agreements fix the reservations of computational resources and the network links required for data transfers. **Figure 6.2** depicts the overall environment with the communications channels for the negotiation and the adapters. The reservations have several objectives:

- Leverage load balancing by distributing the data equally between the computational resources (as far as possible) taking into account the number of nodes available.
- Synchronise the network reservations for the file transfers of the stage-in and stage-out phases so that the input data is available before the docking simulation starts up and the output data is transferred to the users environment right after the end of the docking simulation.
- Guarantee end-to-end connectivity with the required bandwidth.

Through these objectives a number of different co-allocation scenarios can be realised, one of the described here is supporting the WISDOM use-case.

The experiments made proved that the approach is valid and results in a synchronised schedule for the workflow and the network links enabling the High Throughput Virtual Screening.

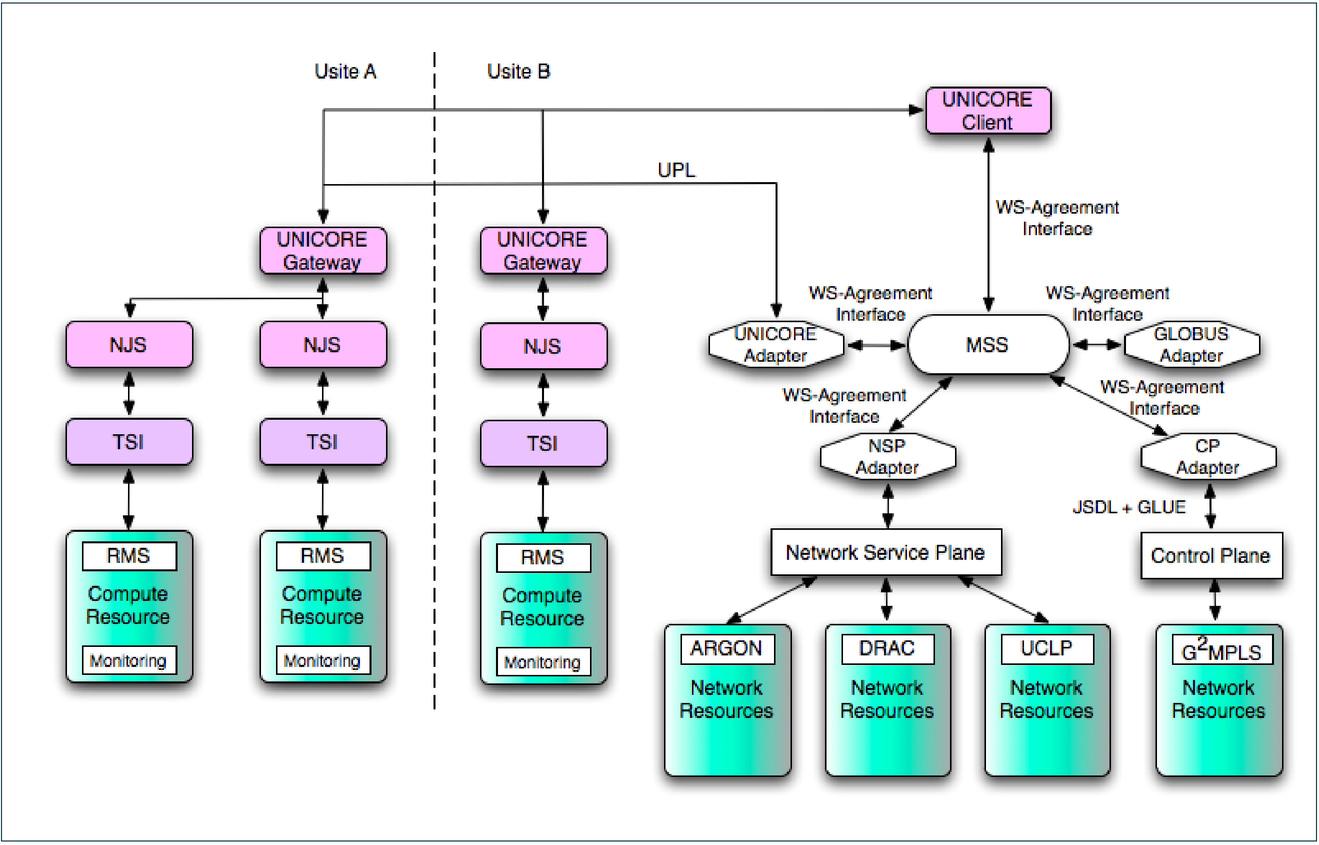

Figure 6.2: Environment for co-allocation of computational resources and network resources

7 Integration between Control Plane and middleware and applications

This section provides a description of the tests carried out to demonstrate the interoperation of the GRID Middleware (Unicore6) system implemented in WP3 with G^{2}MPLS implemented in WP2. The tests have been carried out to demonstrate G^{2}MPLS functionalities triggered by a specific application, in this case KoDaVIS, which needs network connectivity to connect application clients and servers.

7.1 Testing environment and test-bed setup

The environment used for the integration tests includes a G^{2}MPLS domain consisting of four low-cost Ethernet switches and three client nodes. One of these nodes deploys a KoDaVIS client and an instance of the Unicore6 Middleware while the other two nodes deploy both KoDaVIS cooperation and data servers and one of them also a second KoDaVIS client.

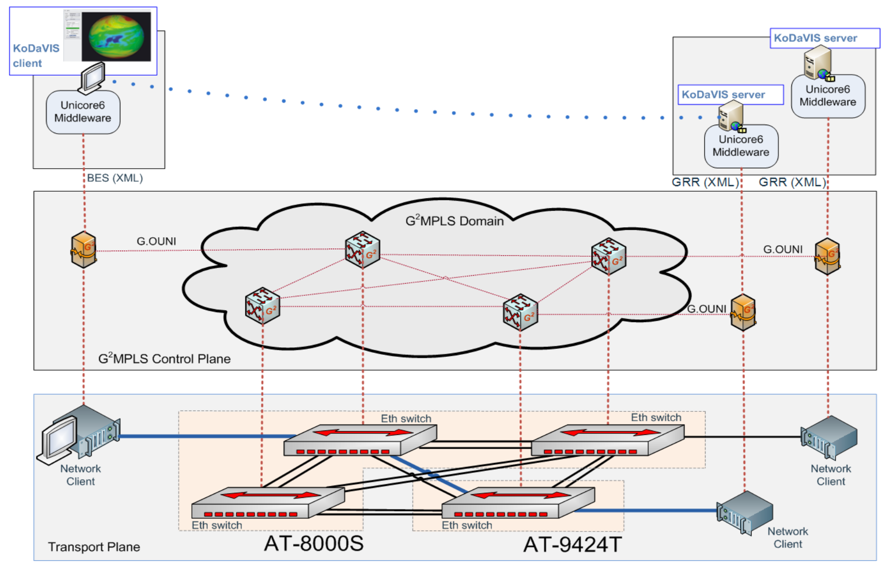

Figure 7.1: Test-bed topology

Each of the client controllers runs the GRID.UNI Gateway (GUNI-GW) module, interconnecting the Middleware instances and the Client Connection Controller (CCC) part of G^{2}MPLS. This module implements two different Web Service servers, namely Basic Execution Service (BES) and GRID Resource Registry (GRR) Service. The former is used for signalling purposes and the later for routing GRID information about the servers. The configuration of the scenario allows the creation of a connection between KoDaVIS client and a KoDaVIS server accepting requests directly from the application and forwarding it to the control plane. The logical layout of the control plane can be seen in Figure 7.2.

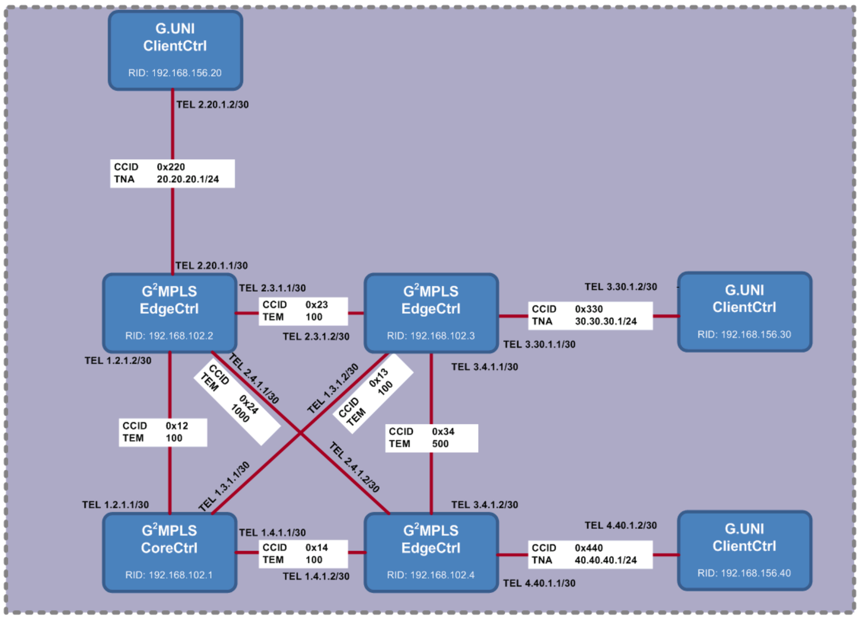

Figure 7.2: Control Plane logical layout

7.2 Functional tests

The functional tests have been focused in the correct operation of the GUNI-GW module. GUNI-GW GRR and BES Web service implementations allow the GRID middleware to request the provisioning of GRID and Network resources in a seamless way and publish information of GRID resources. Besides, GUNI-GW also implements the G^{2}MPLS CORBA methods that trigger the call setup and the OSPF flooding messages.

We have divided the tests in two main blocks, routing and signalling tests. Routing tests are related with the implementation of the GRR Web Service and Signalling tests are related with the implementation of the BES Web Service.

The actual G^{2}MPLS operation has already been described and tested previously and thus is out of the scope of the description of these tests.

7.2.1 Routing

7.2.1.1 *Publish server availability*

Routing tests basically demonstrate the feasibility of flooding GRID information that characterizes the application server. GRR Web Service supports the publication of description documents, in this case glue schema documents that contain information about the server capabilities. Unicore6 publishes the capabilities of each KoDaVIS server (e.g. FreeJobSlots) to the network. The G^2MPLS routing layer handles the information published by the middleware and floods it to all network nodes. The information about KoDaVIS server capabilities is available then at the network clients attached to the test-bed.

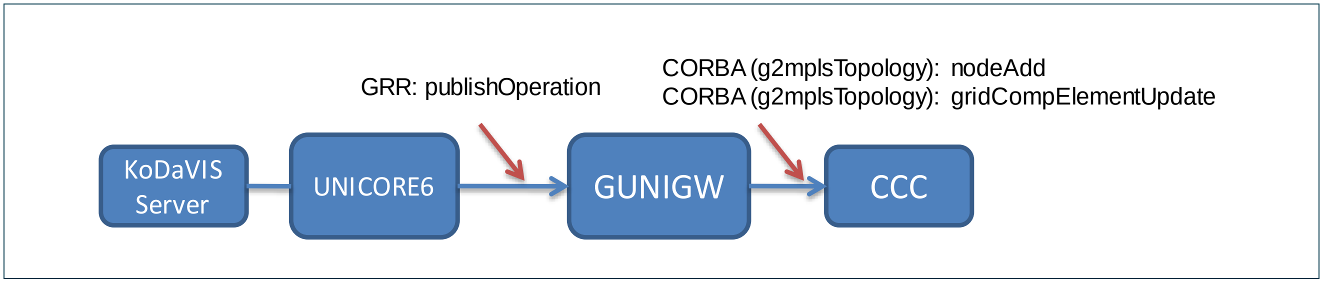
GRR: publishOperation
CORBA (g2mplsTopology): nodeAdd
CORBA (g2mplsTopology): gridCompElementUpdate
KoDaVIS Server
UNICORE6
GUNIGW
CCC

Figure 7.3: Routing message flow

For the sake of simplicity, just one parameter has been used to characterize KoDaVIS servers, as is FreeJobSlots, included in the glue schema. Unicore6 uses XML to send the GRR publishOperation message and a server identifier. GUNI-GW parses the SOAP message and maps the information received into CORBA calls to enable the flooding of the GRID information through the network. The log shows the correct reception of FreeJobSlots parameter and the call of the CORBA methods:

```
2009/05/22 12:29:38 GUNIGW: [DBG] GRR Call: publishOperation (IN)
2009/05/22 12:29:38 GUNIGW: [DBG] Site->Name: KoDaVIS_Srv_1
2009/05/22 12:29:38 GUNIGW: [DBG] Site->UniqueID: 1
2009/05/22 12:29:38 GUNIGW: [DBG] Site->Cluster->Name: KoDaVIS_Srv_1_1
2009/05/22 12:29:38 GUNIGW: [DBG] Site->Cluster->UniqueID: 11
2009/05/22 12:29:38 GUNIGW: [DBG] Site->Cluster->ComputingElement->UniqueID: 111
2009/05/22 12:29:38 GUNIGW: [DBG] Site->Cluster->ComputingElement->GRAMVersion: UNICORE6
2009/05/22 12:29:38 GUNIGW: [DBG] Site->Cluster->ComputingElement->HostName: server1
2009/05/22 12:29:38 GUNIGW: [DBG] Site->Cluster->ComputingElement->GateKeeperPort: 443
2009/05/22 12:29:38 GUNIGW: [DBG] Site->Cluster->ComputingElement->FreeJobSlots: 10
2009/05/22 12:29:38 GUNIGW: [DBG] Sent nodeAdd through CORBA Mgmt interface
2009/05/22 12:29:38 GUNIGW: [DBG] Sent GRIDCompElemUpdate through CORBA Mgmt interface
2009/05/22 12:29:38 GUNIGW: [DBG] GRR Call: publishOperation (OUT)
```

7.2.2 Signalling

Signalling tests consider three possible demonstration scenarios based on decision point for the allocation of resources in the test-bed, unicast, anycast (overlay) or anycast (integrated). BES Web Service is used to send the requests from the middleware to GUNI-GW including a complete Job description by providing jsdl schema documents that contain complete information about the Job request. This request is then forwarded to the G^{2}MPLS CCC in shape of CORBA methods to setup the call and the LSP.

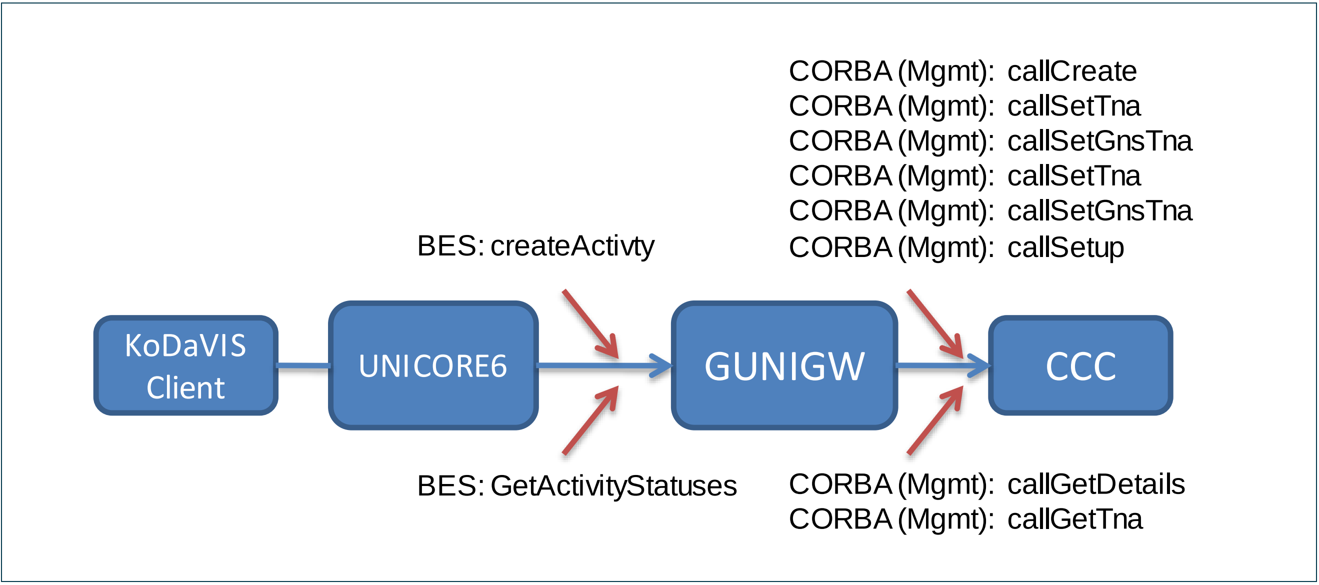

Figure 7.4: Signalling message flow

7.2.2.1 *Unicast Connection Request*

In this scenario, KoDaVIS client knows the tna address of the remote server (Egress TNA) and sends this information to GUNI-GW together with other relevant parameters such as ingress tna, bandwidth, calendar, required FreeJobSlots and a Job identifier. Unicore6 sends a request correctly to the GUNI-GW URL, it parses the XML message and triggers the CORBA methods that start the call signalling, shown in the following log:

```
2009/05/22 12:32:41 GUNIGW: [DBG] WSAG Call: CreateActivity
2009/05/22 12:32:41 GUNIGW: [DBG] Job_ID: 1
2009/05/22 12:32:41 GUNIGW: [DBG] ApplicationName: KoDaVIS
2009/05/22 12:32:41 GUNIGW: [DBG] IngressTNA: 20.20.20.1
2009/05/22 12:32:41 GUNIGW: [DBG] Unicast call
2009/05/22 12:32:41 GUNIGW: [DBG] EgressTNA: 30.30.30.1
2009/05/22 12:32:41 GUNIGW: [DBG] Bandwidth: 50000000
2009/05/22 12:32:41 GUNIGW: [DBG] TotalCPUCount (LB) = 10
2009/05/22 12:32:41 GUNIGW: [DBG] TotalCPUCount (UB) = 10
```

```
2009/05/22 12:32:41 GUNIGW: [DBG] Going to create a GUNIGW CALL with localID 1
2009/05/22 12:32:41 GUNIGW: [DBG] Sent callCreate through CORBA Mgmt interface
2009/05/22 12:32:41 GUNIGW: [DBG] Created call with identity [OPERATOR SPECIFIC]
2009/06/08 12:32:41 GUNIGW: [DBG]         - srcAddr             : (IPv4) 192.168.156.20/32
2009/06/08 12:32:41 GUNIGW: [DBG]         - localId             : 0x0000000000000001
2009/05/22 12:32:41 GUNIGW: [DBG] Sent callSetTna for iTNA through CORBA Mgmt interface
2009/05/22 12:32:41 GUNIGW: [DBG] Sent callSetGnsTna for iTNA through CORBA Mgmt interface
2009/05/22 12:32:41 GUNIGW: [DBG] Sent callSetTna for eTNA through CORBA Mgmt interface
2009/05/22 12:32:41 GUNIGW: [DBG] Sent callSetGnsTna for eTNA through CORBA Mgmt interface
2009/05/22 12:32:41 GUNIGW: [DBG] Sent callSetUp through CORBA Mgmt interface
2009/05/22 12:32:41 GUNIGW: [DBG] WSAG Call: Returning CreateActivity
2009/05/22 12:32:42 GUNIGW: [DBG] Received callRaiseMsgEvent(SetupIndication) CORBA EW interface
2009/05/22 12:32:42 GUNIGW: [DBG] Sent callRaiseMsgEvent(SetupConfirm) through CORBA EW interface
2009/05/22 12:32:42 GUNIGW: [DBG] WSAG Call: GetActivityStatuses
2009/05/22 12:32:42 GUNIGW: [DBG] Requesting details for Call with localID 1 and src LSR (IPv4)
192.168.156.20/32
2009/05/22 12:32:42 GUNIGW: [DBG] Sent callGetDetails through CORBA Mgmt interface
2009/05/22 12:32:42 GUNIGW: [DBG] Sent callGetTna through CORBA Mgmt interface
2009/05/22 12:32:42 GUNIGW: [DBG] operstate: UP
2009/05/22 12:32:42 GUNIGW: [DBG] WSAG Call: Returning GetActivityStatuses
```

7.2.2.2 *Anycast Connection Request (Overlay)*

In this scenario, the MSS (Unicore6) uses the information retrieved from the network, chooses a better KoDaVIS server to handle the KoDaVIS client request, and requests the unicast connection in the G^{2}MPLS Control Plane. Since the server is chosen by the MSS, the interaction between Unicore6 and GUNI-GW is the same as in the unicast scenario.

7.2.2.3 *Anycast Connection Request (Integrated)*

In this anycast scenario, Unicore6 forwards the KoDaVIS client request to the network without specifying the egress TNA and lets the G^{2}MPLS Control Plane to take a decision about the most appropriate KoDaVIS server based on the FreeJobSlots parameter. Then, the control plane proceeds with a setup of the path in the Transport Plane and sends back to the middleware the identifier of the selected KoDaVIS server (candHost).

```
2009/05/22 12:36:28 GUNIGW: [DBG] WSAG Call: CreateActivity
2009/05/22 12:36:28 GUNIGW: [DBG] Job_ID: 1
2009/05/22 12:36:28 GUNIGW: [DBG] ApplicationName: KoDaVIS
2009/05/22 12:36:28 GUNIGW: [DBG] IngressTNA: 20.20.20.1
2009/05/22 12:36:28 GUNIGW: [DBG] Anycast call
2009/05/22 12:36:28 GUNIGW: [DBG] Bandwidth: 80000000
2009/05/22 12:36:28 GUNIGW: [DBG] TotalCPUCount (LB) = 10
2009/05/22 12:36:28 GUNIGW: [DBG] TotalCPUCount (UB) = 10
```

```
2009/05/22 12:36:28 GUNIGW: [DBG] Going to create a GUNIGW CALL with localID 1
2009/05/22 12:36:28 GUNIGW: [DBG] Sent callCreate through CORBA Mgmt interface
2009/05/22 12:36:28 GUNIGW: [DBG] Created call with identity [OPERATOR SPECIFIC]
2009/06/08 12:36:28 GUNIGW: [DBG]         - srcAddr             : (IPv4) 192.168.156.20/32
2009/06/08 12:36:28 GUNIGW: [DBG]         - localId             : 0x0000000000000001
2009/05/22 12:36:28 GUNIGW: [DBG] Sent callSetTna for iTNA through CORBA Mgmt interface
2009/05/22 12:36:28 GUNIGW: [DBG] Sent callSetGnsTna for iTNA through CORBA Mgmt interface
2009/05/22 12:36:28 GUNIGW: [DBG] Sent callSetTna for eTNA through CORBA Mgmt interface
2009/05/22 12:36:28 GUNIGW: [DBG] Sent callSetGnsTna for eTNA through CORBA Mgmt interface
2009/05/22 12:36:28 GUNIGW: [DBG] Sent callSetUp through CORBA Mgmt interface
2009/05/22 12:36:29 GUNIGW: [DBG] WSAG Call: Returning CreateActivity
2009/05/22 12:36:29 GUNIGW: [DBG] Received callRaiseMsgEvent(SetupIndication) CORBA EW interface
2009/05/22 12:36:29 GUNIGW: [DBG] Sent callRaiseMsgEvent(SetupConfirm) through CORBA EW interface
2009/05/22 12:36:36 GUNIGW: [DBG] WSAG Call: GetActivityStatuses
2009/05/22 12:36:36 GUNIGW: [DBG] Requesting details for Call with localID 1 and src LSR
192.168.156.20
2009/05/22 12:36:36 GUNIGW: [DBG] Sent callGetDetails through CORBA Mgmt interface
2009/05/22 12:36:36 GUNIGW: [DBG] Sent callGetTna through CORBA Mgmt interface
2009/05/22 12:36:36 GUNIGW: [DBG] operstate: UP
2009/05/22 12:36:36 GUNIGW: [DBG] CandHost: 1
2009/05/22 12:36:36 GUNIGW: [DBG] WSAG Call: Returning GetActivityStatuses
```

7.2.2.4 *Connection Termination*

This test is common to the previous described scenarios. Unicore6 forwards the KoDaVIS client termination request to GUNI-GW specifying the Job ID, and the corresponding *callDestroy* CORBA call is triggered accordingly as shown in the log:

```
2009/05/22 12:45:07 GUNIGW: [DBG] WSAG Call: TerminateActivities
2009/05/22 12:45:07 GUNIGW: Going to destroy a GUNIGW CALL with localID 1
2009/05/22 12:45:07 GUNIGW: Sent callDestroy through CORBA Mgmt interface
2009/05/22 12:45:07 GUNIGW: WSAG Call: Returning TerminateActivities
```

7.2.2.5 *Connection request failure*

When G^2MPLS is unable to establish the requested connection due to lack of resources, failure in the configuration of devices, timeouts or other general errors, the MSS is informed with an exception:

```
2009/06/11 11:52:44 GUNIGW: [DBG] WSAG Call: CreateActivity
2009/06/11 11:52:44 GUNIGW: [DBG] Job_ID: 1
2009/06/11 11:52:44 GUNIGW: [DBG] ApplicationName: KoDaVis
2009/06/11 11:52:44 GUNIGW: [DBG] IngressTNA: 20.20.20.1
2009/06/11 11:52:44 GUNIGW: [DBG] Anycast call
```

```
2009/06/11 11:52:44 GUNIGW: [DBG] Bandwidth: 800000000
2009/06/11 11:52:44 GUNIGW: [DBG] TotalCPUCount: LB=10
2009/06/11 11:52:44 GUNIGW: [DBG] TotalCPUCount: UB=10
2009/06/11 11:52:44 GUNIGW: [DBG] Bandwidth2: 1211922944
2009/06/11 11:52:44 GUNIGW: [DBG] Going to create a GUNIGW CALL with localID 1
2009/06/11 11:52:44 GUNIGW: [DBG] Sent callCreate through CORBA Mgmt interface
2009/06/11 11:52:44 GUNIGW: [DBG] Created call with identity [OPERATOR SPECIFIC]
2009/06/11 11:52:44 GUNIGW: [DBG]        - srcAddr            : (IPv4) 192.168.156.20/32
2009/06/11 11:52:44 GUNIGW: [DBG]        - localId            : 0x0000000000000001
2009/06/11 11:52:44 GUNIGW: [DBG] Sent callSetTna for iTNA through CORBA Mgmt interface
2009/06/11 11:52:44 GUNIGW: [DBG] Sent callSetGnsTna for eTNA through CORBA Mgmt interface
2009/06/11 11:52:44 GUNIGW: [DBG] Sent callSetUp through CORBA Mgmt interface
2009/06/11 11:53:14 GUNIGW: [DBG] Timeout!
2009/06/11 11:53:14 GUNIGW: [DBG] callSetup through CORBA Mgmt interface failed for unknown reasons
2009/06/11 11:53:14 GUNIGW: [DBG] Sending exception…
2009/06/11 11:52:44 GUNIGW: [DBG] WSAG Call: Returning CreateActivity
```

7.3 Test Conclusions

All the performed tests have successfully validated the interoperation of MSS and G^{2}MPLS by means of the GUNI-GW. With this module, any application is able to request job requests or submit glue schema documents to G^{2}MPLS by using the BES and GRR Web Services. The tests have proved the designed routing and signalling functionalities towards the integration of seamless GRID and network resource management procedures.

8 Integration of security elements into Network Resource Provisioning Systems

This section introduces the tests realized in order to achieve WP1-WP4 integration. It is divided into three sub-sections. First and second sections present how the Authentication and Authorisation Infrastructure (AAI) has been deployed in both real and virtual test-beds; while third sections presents all the JUnit tests done in order to assure that the security library has the expected behaviour.

8.1 Real test-bed

The real test-bed has suffered a huge growth in the last months. Currently, there are more than 10 domains involved in the Harmony test-bed. These domains are located in different countries all over the world, such as Spain, Germany, the Netherlands, United Kingdom and Poland in Europe and Canada, Korea and the United States of America in the rest of the world. Due to the major works to be done in the whole test-bed for the AAI deployment in Harmony, the plan for deploying security in the real test-bed has consisted in adding a new Point of Entry (PoE) to the system that will work in a secure way. Thus, new secure-IDB and secure-GUI have been configured and deployed. This secure-IDB represents a new top level in the Harmony hierarchy currently deployed in the real test-bed and requires signed and encrypted messages from users. As for the secure-GUI, it has also been deployed in order to communicate with the new secure-IDB.

8.2 Virtual test-bed

The virtual test-bed in WP1 has been being used for deploying and testing all the new developments done during last phases of the PHOSPHORUS project. The virtual test-bed is named this way because the physical equipment is not present, that is, no transport networks are being controlled by the NRPSs in the virtual test-bed. The NRPSs involved in the virtual test-bed work in mock mode; that is to say, they emulate the physical network device behaviour. The virtual test-bed is composed of six Inter-Domain Brokers (IDBs), each one controlling one Harmony NRPS adapter, each adapter controlling an NRPS in mock mode. Next figure depicts the topology of the control and service planes in Harmony's virtual test-bed:

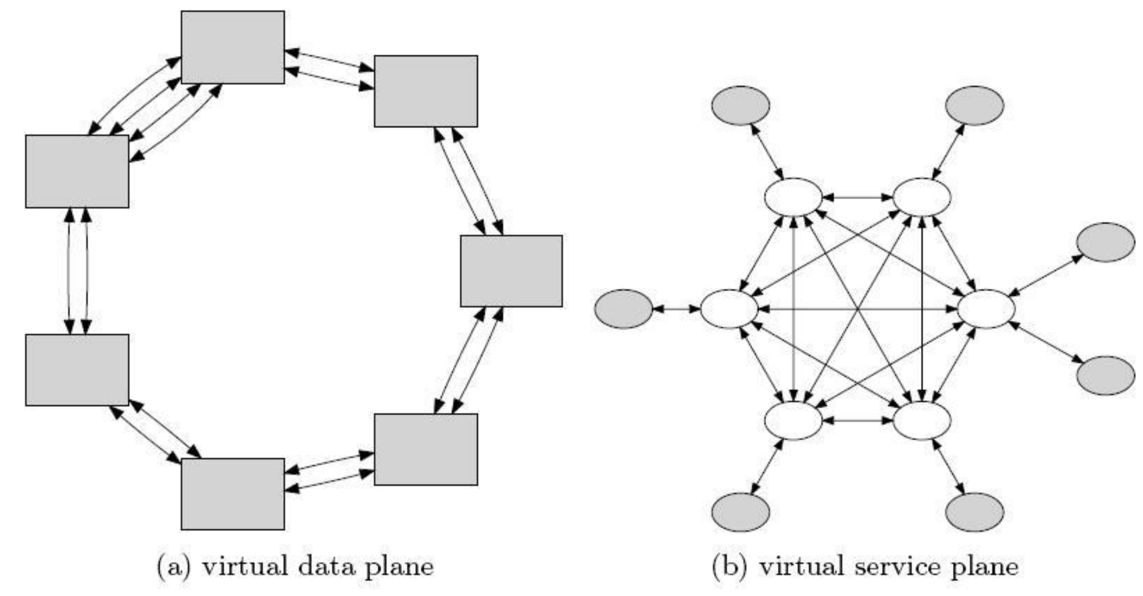

Figure 8.1: Virtual test-bed in Harmony: Data plane and Service plane

The IDBs located in the virtual test-bed are working in distributed mode (or peer-to-peer approach). Thus, each IDB floods to its peers the topology information present in its database. The flooding process follows an OSPF-like behaviour [PHOSPHORUS-D1.8]. The IDBs are distributed between Canada, Germany and Spain. The whole virtual test-bed has an AAI integrated. In this sense, each entity belonging to the virtual test-bed (GUI, IDB, or HNA) has the security modules enabled and configured in order to allow only communication between these entities.

```
Keystore type: jks
Keystore provider: SUN

*******************************************
*******************************************


Alias name: 10.1.3.200_8080_harmony-idb-2
Creation date: 26/05/2009
Entry type: trustedCertEntry

Owner: CN=10.1.3.200_8080_harmony-idb-2, O=Internet Widgits Pty Ltd, ST=Some-State, C=AU
Issuer: CN=10.1.3.200_8080_harmony-idb-2, O=Internet Widgits Pty Ltd, ST=Some-State, C=AU
Serial number: b487520fe800e72b
Valid from: Tue May 26 11:33:56 CEST 2009 until: Fri May 25 11:33:56 CEST 2012
Certificate fingerprints:
         MD5:  98:BB:B3:81:F7:55:FD:CE:DB:CF:CB:EF:22:61:4F:BD
         SHA1: C6:D2:37:88:87:60:C2:3F:02:FE:62:4D:77:DF:43:CC:90:90:D3:67
```

Code 8.1: Example of the certificate of one IDB

Each entity present in the service plane has its own certificate. The configuration of the AAI has consisted of exchanging the certificates between entities (manually, no automatic protocol implemented at the moment) in order to allow each entity to authenticate the sender of the request.

8.3 JUnit tests

This section presents the JUnit tests done in order to test the integration process between Harmony and WP4 developments. For each module, the table depicts each tests realized and it also presents the description of each test and the possible behaviour while the execution of the test.

Module	Test	Description
AuthN	`testEncryptedCreate`	Tests the creation of a reservation with the encryption enabled. Returns the reservation ID if the system behaviours as expected, fails otherwise.
	`testEncryptedSignedCreate`	Tests the creation of a reservation with the encryption and signature of the message enabled. Returns the reservation ID if the system behaviours as expected; fails otherwise.
	`testSignedCreate`	Tests the creation of a reservation with the message signed. Returns the reservation ID if the process ends successfully; fails otherwise
Encryption	`testEncryptSignUnsignedDescrypt`	Checks the workflow of encrypt, sign, unsign, and decrypt the message. Fails if the message decrypted is not the same as the initial message. Success otherwise.
	`testGetRequest`	Tests getting the requests from one encrypted message. Fails if the request obtained differs from the initial one. Success otherwise.
	`testGetResponse`	Tests getting the encrypted response from a response message. Success if the response is decrypted correctly. Fails otherwise.
	`testGetUserID`	Tests getting the user ID from one previously encrypted message. Success if the user ID is decrypted correctly. Fails otherwise.
GAAAtk	`testGeneralAPI`	Tests the most general PEP API. It has to be adopted for XACML-NRPS profile and using topology exchange. Returns true if the topology method is allowed. Fails otherwise.
	`testSimpleAPI`	Tests a simpler PEP API.

	`testTnaBasedPolicy`	
HarmonyPEP	`testAuthZInValidCreateReservation`	Tests authorization while trying to create a reservation. PEP returns invalid permissions to create a reservation
	`testAuthZInValidIsAvailable`	Tests authorization while making an isAvailable request. PEP returns invalid permissions for this type of request.
	`testAuthZValidCancelJob`	Tests authorization while cancelling one job. PEP checks the permissions and authorizes the cancellation.
	`testAuthZValidCreateReservation`	Tests authorization while creating a reservation. PEP checks the permissions and authorizes the creation of one advance reservation. Returns the reservation ID.
	`testAuthZValidIsAvailable`	Tests authorization while sending an isAvailable request. PEP checks the permissions and authorizes the request. Returns whether the endpoints requested are available or not.
	`testAuthZCompleteValidPEP`	Tests the whole workflow when creating and cancelling a reservation. PEP checks the permissions and acts depending on the permissions passed to the test.
HarmonyTVS	`testTVS`	Tests the authorization module using PEP and TVS. It checks whether the token created is authorized or not for a concrete request (create reservation)
Utils	`testExtractTokenValue`	Tests the process of extracting the token from the XML message.
	`testFormatGRI`	Tests whether the format of the GRI is correct or not
	`testInterDomainManagToken`	Tests the inter-domain token manager. This method tests the exchange of token between two different Network Service Plane entities.
	`testManagementToken`	Tests the token manager. This method tests the main functionalities of the token manager.

Table 8.1: JUnit tests for WP1-WP4 integration

8.4 Conclusions

The tests realized have proved that the integration between the developments of WP4 and developments of WP1 has been achieved. This integration represented the main objective of the collaboration between both work packages, since it was the way of adding security within the Network Service Plane and it also was the way of demonstrating that developments of WP4 were working and can be integrated into one real system.

9 Integration of security elements into G^{2}MPLS Control Plane

This section discusses the final architecture and the tests executed to validate the integration of the GAAA-TK library into the G^{2}MPLS Control Plane. In this test scenario the GAAA-TK was contributed by WP4 through D4.3.1/D4.5 and the G^{2}MPLS stack from WP2 through the D2.10.

The tests have been executed successfully on the same multi-domain G^{2}MPLS testbed described Section 2 and impacted just the G^{2}MPLS call signalling phase.

9.1 AuthN/AuthZ support in G^{2}MPLS

The AuthN/AuthZ model adopted in G^{2}MPLS Control Plane is based on the GAAA AuthZ architecture for Optical Network Resource Provisioning as described in [**PHOSPHORUS-D2.8** and **PHOSPHORUS-D4.3.1**]. The G^{2}MPLS control plane adopts the *Chain reservation sequence* model, in which the user contacts only the local network domain and the control plane signalling is in charge of triggering the resource reservation authorization in each consecutive domain.

G^{2}MPLS uses AuthZ pilot tokens, which contain a brief description of some context information, are valid end-to-end and can be used to set up a multi-domain TVS infrastructure for distributed token-based access control. The complex architecture of the multi-domain distributed GAAA/AuthZ framework is completely transparent to G^{2}MPLS, i.e. how the context information is generated and made valid in different domains.

According to this approach, the Policy Enforcement Point (PEP) located in each domain remains a stateless entity that does not store any information about the associations between existing sessions and the active tokens: the PEP just handles AuthN/AuthZ requests, forwards them to the GAAA layer to get a pilot token and, finally, forward the token back to the signalling entities for its (multi-domain) propagation (ref. **Figure 9.1**).

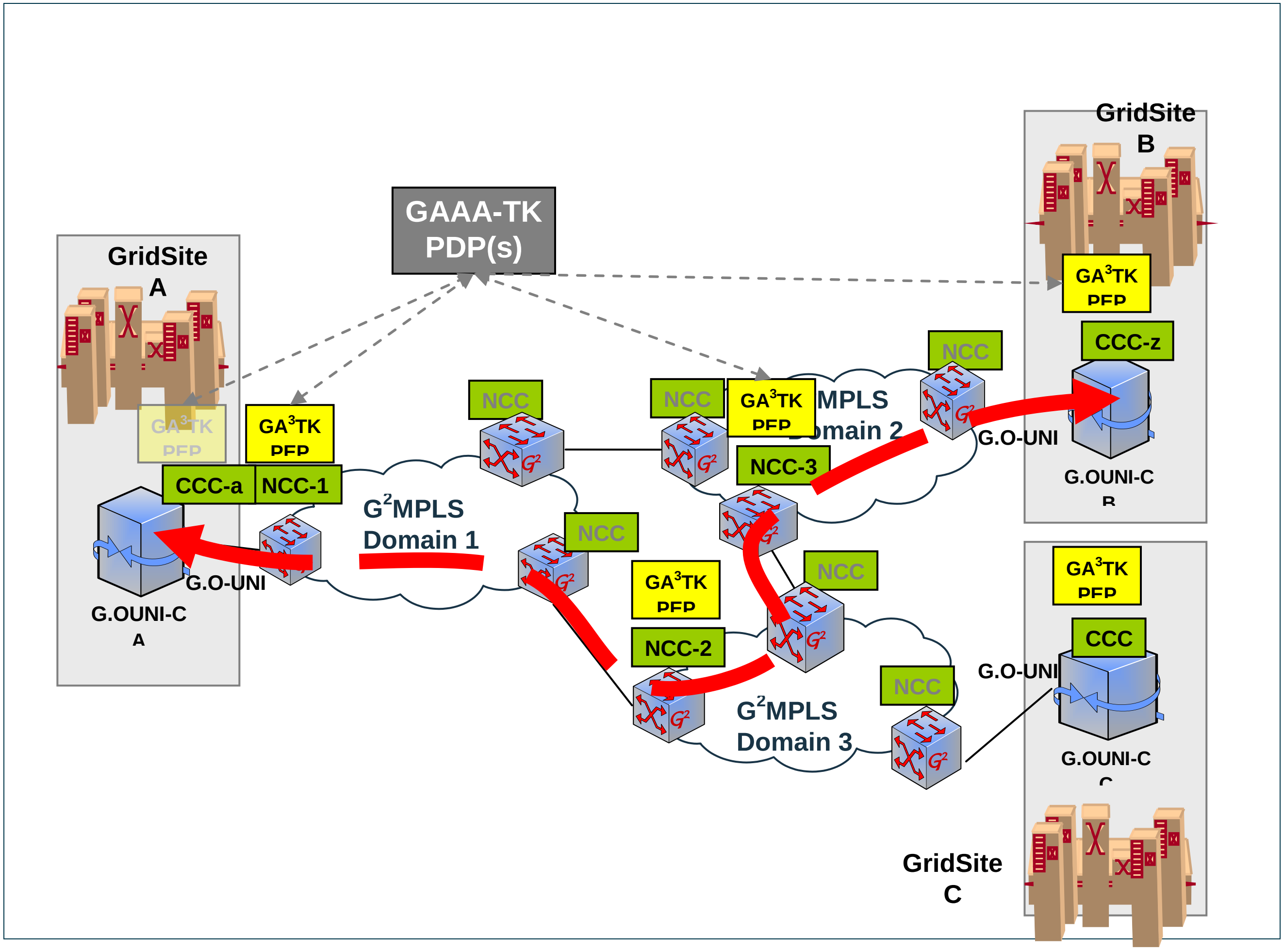

Figure 9.1: G^{2}MPLS inter-domain signalling with AuthN/AuthZ support.

9.2 Tests on AuthN/AuthZ functionalities in the PHOSPHORUS G^{2}MPLS multi-domain testbed

With reference to the multi-domain G^{2}MPLS testbed described Section 2 (PSNC+UESSEX local testbeds), the main G^{2}MPLS modules involved in these specific tests have been:

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

  - GAAA-TK
- in the egress G^{2}MPLS UNI client
  - Client Call Controller (CCC-z)
  - PEP-GW
  - GAAA-TK

The actions on the transit/egress domain and the egress client are similar and just token-based. On the contrary the actions on the ingress domain are based on context information regarding the call and tokens. In details,

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

The following logs describe the successful interactions between PEPGW and GAAA-TK in the reference signalling scenarios described in Section 2.2.3

PEP-GW-1: issues *authorizeAction*

```
Mon May 04 17:31:55 CEST 2009 === PEP GW started
Mon May 04 17:31:59 CEST 2009 === [AAA Server] Received a request for a new token...
Mon May 04 17:31:59 CEST 2009 === [RequestProcessing] Processing request...
Mon May 04 17:31:59 CEST 2009 === [RequestProcessing] RequestType = 1
Mon May 04 17:31:59 CEST 2009 === [RequestProcessing] Action = access
Mon May 04 17:31:59 CEST 2009 === [RequestProcessing] SubjectID = admin@g2mpls.testbed
Mon May 04 17:31:59 CEST 2009 === [RequestProcessing] Source = 10.10.10.1
Mon May 04 17:31:59 CEST 2009 === [RequestProcessing] Target = 20.20.20.1
Mon May 04 17:31:59 CEST 2009 === [RequestProcessing] DomainID = http://testbed.ist-phosphorus.eu
Mon May 04 17:31:59 CEST 2009 === [RequestProcessing] Credentials datastructures built
Mon May 04 17:32:00 CEST 2009 === [RequestProcessing] Request authorized
```

PEP-GW-1: issues *getXMLToken*

```
Mon May 04 17:32:00 CEST 2009 === [RequestProcessing] SessionID created: 4d2a17e23e18187d
Mon May 04 17:32:00 CEST 2009 === [RequestProcessing] Token built
```

GAAA-TK-1: on *authorizeAction*

```
Policy file PolicyResolver'd =
data/policy/nrp/testbed.ist-phosphorus.eu/g2mpls-policy-testbed-demo001.xml

XACMLPDP Request received by XACMLPDPsimple.requestPDP (RequestCtx,
PolicyRef)):
<Request>
<Subject
SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
<Attribute
AttributeId="http://authz-interop.org/AAA/xacml/subject/subject-role"
DataType="http://www.w3.org/2001/XMLSchema#string"
Issuer="http://testbed.ist-phosphorus.eu/phosphorus/aaa/AttributeIssuer"
IssueInstant="2009-05-
04T17:31:59.893000000+02:00"><AttributeValue>admin</AttributeValue></Attribute>
<Attribute
AttributeId="http://authz-interop.org/AAA/xacml/subject/subject-confdata"
DataType="http://www.w3.org/2001/XMLSchema#string"
Issuer="http://testbed.ist-phosphorus.eu/phosphorus/aaa/AttributeIssuer"
IssueInstant="2009-05-
04T17:31:59.893000000+02:00"><AttributeValue>IGhA11vwa8bUktYhuU9que+d4XLUvJFHrtDC/OE3Ui1bxtmuCxLldw
==</AttributeValue></Attribute>
<Attribute
AttributeId="http://authz-interop.org/AAA/xacml/subject/subject-context"
DataType="http://www.w3.org/2001/XMLSchema#string"
Issuer="http://testbed.ist-phosphorus.eu/phosphorus/aaa/AttributeIssuer"
IssueInstant="2009-05-
04T17:31:59.893000000+02:00"><AttributeValue>demo001</AttributeValue></Attribute>
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="http://www.w3.org/2001/XMLSchema#string"
Issuer="http://testbed.ist-phosphorus.eu/phosphorus/aaa/AttributeIssuer"
IssueInstant="2009-05-
04T17:31:59.893000000+02:00"><AttributeValue>WHO740@users.collaboratory.nl</AttributeValue></Attrib
ute>
</Subject>
<Resource>
<Attribute
AttributeId="http://authz-interop.org/AAA/xacml/resource/resource-domain"
DataType="http://www.w3.org/2001/XMLSchema#string"
```

```
Issuer="http://testbed.ist-phosphorus.eu/phosphorus/aaa/AttributeIssuer"
IssueInstant="2009-05-
04T17:31:59.873000000+02:00"><AttributeValue>g2mpls</AttributeValue></Attribute>
<Attribute
AttributeId="http://authz-interop.org/AAA/xacml/resource/source"
DataType="http://www.w3.org/2001/XMLSchema#string"
Issuer="http://testbed.ist-phosphorus.eu/phosphorus/aaa/AttributeIssuer"
IssueInstant="2009-05-
04T17:31:59.873000000+02:00"><AttributeValue>10.7.12.2</AttributeValue></Attribute>
<Attribute
AttributeId="http://authz-interop.org/AAA/xacml/resource/target"
DataType="http://www.w3.org/2001/XMLSchema#string"
Issuer="http://testbed.ist-phosphorus.eu/phosphorus/aaa/AttributeIssuer"
IssueInstant="2009-05-
04T17:31:59.873000000+02:00"><AttributeValue>10.3.17.3</AttributeValue></Attribute>
<Attribute
AttributeId="http://authz-interop.org/AAA/xacml/resource/resource-realm"
DataType="http://www.w3.org/2001/XMLSchema#string"
Issuer="http://testbed.ist-phosphorus.eu/phosphorus/aaa/AttributeIssuer"
IssueInstant="2009-05-04T17:31:59.873000000+02:00"><AttributeValue>testbed.ist-
phosphorus.eu</AttributeValue></Attribute>
<Attribute
AttributeId="http://authz-interop.org/AAA/xacml/resource/resource-type"
DataType="http://www.w3.org/2001/XMLSchema#string"
Issuer="http://testbed.ist-phosphorus.eu/phosphorus/aaa/AttributeIssuer"
IssueInstant="2009-05-
04T17:31:59.873000000+02:00"><AttributeValue>testbed</AttributeValue></Attribute>
<Attribute
AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
DataType="http://www.w3.org/2001/XMLSchema#anyURI"
Issuer="http://testbed.ist-phosphorus.eu/phosphorus/aaa/AttributeIssuer"
IssueInstant="2009-05-04T17:31:59.873000000+02:00"><AttributeValue>http://testbed.ist-
phosphorus.eu/g2mpls/testbed</AttributeValue></Attribute>
</Resource>
<Action>
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string"
Issuer="http://testbed.ist-phosphorus.eu/phosphorus/aaa/AttributeIssuer"
IssueInstant="2009-05-
04T17:31:59.873000000+02:00"><AttributeValue>access</AttributeValue></Attribute>
</Action>
</Request>
```

GAAA-TK-1: on *getXMLToken*

```
PDP Response returned to PEP:
<Response>
<Result ResourceID="http://testbed.ist-phosphorus.eu/g2mpls/testbed">
<Decision>Permit</Decision>
<Status>
<StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
</Status>
</Result>
</Response>

PEPinputParser: Received PDPResponse. Parsing...
```

PEP-GW-n: issues *validateToken*

```
Mon May 04 17:32:00 CEST 2009 === [AAA Server] Received a request for a new token...
Mon May 04 17:32:00 CEST 2009 === [RequestProcessing] Processing request...
Mon May 04 17:32:00 CEST 2009 === [RequestProcessing] RequestType = 2
Mon May 04 17:32:00 CEST 2009 === [RequestProcessing] Token = <AAA:AuthzToken
xmlns:AAA="http://www.aaauthreach.org/ns/AAA" Issuer="http://testbed.ist-phosphorus.eu/aaa/TVS"
SessionId="4d2a17e23e18187d" TokenId="cecfc06e05c4b46ab4c5c36b991babf1">
<AAA:TokenValue>8cf49fa4b23c7910c772e9b14f2eec77f6f70311</AAA:TokenValue>
<AAA:Conditions NotBefore="2009-05-04T15:32:00.476Z" NotOnOrAfter="2009-07-
05T07:32:00.476Z"></AAA:Conditions>
</AAA:AuthzToken>
Mon May 04 17:32:00 CEST 2009 === [RequestProcessing] Request authorized
```

GAAA-TK-n: on *validateToken*

```
Received an authorization request
Token time is valid
TVS validation: Token type = 10
```

10 Integration of security elements into middleware and applications

PHOSPHORUS has a layered architecture spanning from the applications on the top to the network layer on the bottom as depicted in **Figure 10.1:** PHOSPHORUS layered architecture below.

This layered application implies that direct communication and interaction of a component in a certain layer is only possible and desired between components in the layers above and below this layer. With the only exception that applications probably may directly communicate with the NRPS in case there is no GRID middleware or the GRID middleware is not required for the execution of the application.

The credentials of the user are managed through functions of the middleware layer, e.g. the keystore of the UNICORE client. When a user starts creating a new job involving both computational and network resources he authenticates himself vis-à-vis the middleware with his certificate stored previously in the keystore. In the further process UNICORE creates a SAML[2] assertion with the user's distinguished name (DN) taken from the certificate. This SAML assertion is then used internally by the UNICORE system but also transferred to the MetaScheduling Service (MSS) to authenticate the user towards the network layer. To allow the recipient of the SAML assertion (the MSS) to verify that the SAML assertion of the user comes from a trusted entity it is chained with a SAML assertion of the forwarding entity, i.e. the UNICORE system.

The MSS now forwards the chain of SAML assertions to the HARMONY system adding another SAML assertion with his DN. This is done using the client API got the HARMONY services. HARMONY receives a complete chain of SAML assertions from the user's DN down to the MSS' DN, which can be evaluated with respect of integrity and trust.

In case of positive evaluation HARMONY processes the assertion using the GAAA-Library in order to produce the security token for the network layer. The token created then also contains the DN of the user that originally started the creation of the job.

[2] Security Assertion Markup Language

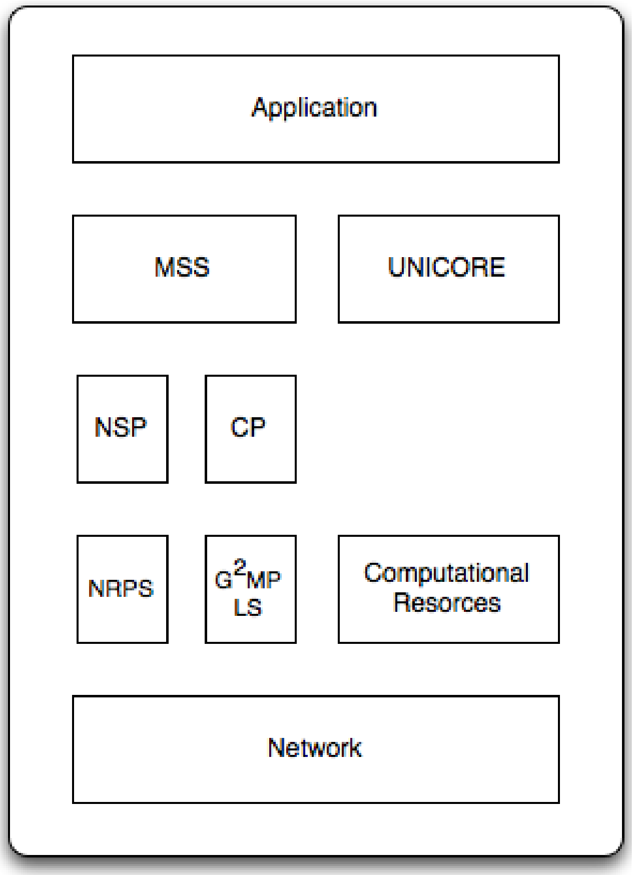

Figure 10.1: PHOSPHORUS layered architecture

11 Conclusions

The deliverable summarizes the results of the tests which were conducted as part of the PHOSPHORUS project. The tests aim to assess the ideas brought by the PHOSPHORUS consortium as well as verify the software developed by the consortium.

The ideas and developments of PHOSPHORUS were tested in a real optical network of European scope with a set of modern scientific applications which make use of the network and the PHOSPHORUS developments. This way the testbed is representative of a modern GRID environment in which demanding applications running on computational nodes use a transmission network to exchange data between the nodes and access external devices.

The tests prove that the PHOSPHORUS ideas can be implemented in real networks and can benefit scientific applications giving them better control over the network and let them utilise the network in a more efficient way. They also prove that the software developed by the consortium is operating properly in a real network with a set of network elements which were selected by the consortium to be supported by the PHOSPHORUS software. The software is available from the PHOSPHORUS consortium and can be deployed by external users in their networks.

12 References

CRISTEA-2007	"The Token Based Switch: Per-Packet Access Authorisation to Optical Shortcuts", by Mihai-Lucian Cristea, Leon Gommans, Li Xu, and Herbert Bos, in Proceedings of IFIP Networking, Atlanta, GA, USA, May 2007
EGEE	www.eu-egee.org
EU FP7 FEDERICA	www.fp7-federica.eu
Globus	www.globus.org
IETF73	www.ietf.org
Internet2	www.internet2.edu
PHOSPHORUS-D1.4	PHOSPHORUS deliverable D1.4: Definition and development of the Network service Plane and northbound interfaces development
PHOSPHORUS-D1.8	PHOSPHORUS deliverable D1.8: NSP enhancements
PHOSPHORUS-D2.1	PHOSPHORUS deliverable D2.1: The Grid-GMPLS Control Plane architecture
PHOSPHORUS-D2.6	PHOSPHORUS deliverable D2.6: Deployment models and solutions of the Grid-GMPLS Control Plane
PHOSPHORUS-D2.8	PHOSPHORUS deliverable D2.8, “Design of the Grid-GMPLS Control Plane to support the Phosphorus Grid AAI”
PHOSPHORUS-D2.10	PHOSPHORUS deliverable D2.10, “Final Grid-GMPLS Control plane prototype”
PHOSPHORUS-D4.3.1	PHOSPHORUS deliverable D4.3.1, “GAAA toolkit pluggable components and XACML policy profile for ONRP”.
PHOSPHORUS-D4.5	PHOSPHORUS deliverable D4.5, “Updated GAAA Toolkit library for ONRP”.
PHOSPHORUS-D6.1	PHOSPHORUS deliverable D6.1: Test-bed design
RFC1918	Address Allocation for Private Internets
Tinc	www.tinc-vpn.org
www1	http://node01.PHOSPHORUS.man.poznan.pl/GRIDftp/
www2	http://node01.PHOSPHORUS.man.poznan.pl/tivoli_fzj/

13 Acronyms

AAA	Authentication, Authorisation and Accounting
AAI	Authentication and Authorisation Infrastructure
API	Application Programming Interface
ARGIA	Ongoing evolution of UCLPv2 towards a commercial product
ARGON	Allocation and Reservation in GRID-enabled Optic Networks
AuthZ	Authorization
BES	Basic Execution Service
Canet	Research and Education Network and Canada
CCC	Calling/Called Party Call Controller
CORBA	Common Object Request Broker Architecture
CP	Control Plane
DN	Distinguished Name
DDSS	Distributed Data Storage System
DRAC	Dynamic Resource Allocation Controller
e2e	end-to-end
EGEE	Enabling GRIDs for E-sciencE (European GRID Project)
E-NNI	External Network to Network Interface
EPR	Endpoint Reference
ERO	Explicit Route Object
Ethernet	A family of technologies for local area computer networks
FSC	Fiber Switching Capable
GAA-TK	Generalized Authentication, Authorization and Accounting Tool Kit
G.E-NNI	GRID enable E-NNI Interface
G.O-UNI	GRID Optical User to Network Interface
GAAA-AuthZ	Generic AAA Authorisation Framework
GÉANT2	Pan-European Gigabit Research Network
gLite	EGEE GRID middleware
GMPLS	Generalized MPLS (MultiProtocol Label Switching)
G^{2}MPLS	GRID-GMPLS (enhancements to GMPLS for GRID support)
G^2.CCC	G^{2}MPLS Client Call Controller

G^2.ENNI-OSPF	G^2MPLS E-NNI OSPF Controller
G^2.ENNI-RSVP	G^2MPLS E-NNI RSVP Controller
G^2.INNI-OSPF	G^2MPLS I-NNI OSPF Controller
G^2.INNI-RSVP	G^2MPLS I-NNI RSVP Controller
G^2.NCC	G^2MPLS Network Call Controller
G^2.OSPF-TE	G^2MPLS OSPF-TE
G^2.PC	G^2MPLS Persistency Controller
G^2.PCE	G^2MPLS Path Computing Element
G^2.RC	G^2MPLS Recovery Controller
G^2.UNI-OSPF	G^2MPLS UNI OSPF Controller
G^2.UNI-RSVP	G^2MPLS UNI RSVP Controller
GRI	Global Reservation Identifier
GRR	GRID Resource Registry
GUI	Graphical User Interface
GUNI-GW	G-UNI Gateway (gateway between UNI client and applications)
Harmony	Optical Network Resource Brokering System
HG^{2}GH	Harmony to G^{2}MPLS Gateway
HNA	Harmony NRPS Adapter
HSI	Harmony Service Interface
HSVO	Health Services Virtual Organization
I-NNI	Inter-Network to Network Interface
IDB	Inter-Domain Broker
IDC	Inter-Domain Controller
INCA	Intelligent Network Caching Architecture
JUnit	Java Testing Framework
KoDaVIS	Tool for Distributed Collaborative Visualisation
L2VPN	Layer 2 Virtual Private Network
LSC	Lambda Switching Capable
LSP	Label Switched Path
MSS	Meta-Scheduling System
NCC	Network Call Controller
NCP	Network Control Plane
NE	Network Element
NREN	National Research and Education Network
NRPS	Network Resource Provisioning System
NSP	Network Service Plane
O-UNI	Optical User to Network Interface
ONL	Optical Networking Laboratory
OSPF	Open Shortest Path First
OSPF-TE	OSPF Traffic Engineering
PEP	Policy Enforcement Point
PoE	Point of Entry
Quagga	Software Routing Suite (www.quagga.net)

RSVP	Resource Reservation Protocol
RSVP-TE	RSVP Traffic Engineering
SAML	Security Assertion Markup Language
SOAP	Simple Object Access Protocol
TBN	Token Based Networking
TBS-IP	Token Based Switch over IP
TOPS	Technology for Optical Pixel Streaming
TNA	Transport Network Assigned Address
TNRC	Transport Network Resource Controller
TSM	Tivoli Storage Manager, a commercial backup/archive software by Tivoli, formerly known as IBM's ADSM)
TVS	Token Validation Service
UCLP	User Controlled Light Paths
UNI	User to Network Interface
UNI-C	User Network Interface – Client side
UNICORE	European GRID Middleware (UNiform Access to COmpute REsources)
VPN	Virtual Private Network
VLS	VideoLAN Client
VO	Virtual Organization
VOMS	Virtual Organization Membership Service
XACML	eXtensible Access Control Markup Language
XML	Extensible Mark-up Language

Disclaimer