Grant Agreement No.: 604590
Instrument: Large scale integrating project (IP)
Call Identifier: FP7-2012-ICT-FI

eXperimental Infrastructures for the Future Internet

D3.4: XIFI Infrastructure Network Adaptation Mechanisms API

Revision: v.1.3

Work package	WP 3
Task	Task 1
Due date	31/03/2014
Submission date	30/04/2014
Deliverable lead	Telefónica I+D (TID)
Authors	Luis M. Contreras (TID), Óscar González (TID), Jaume Marhuenda (TID), Giuseppe Cossù (CNET), José I. Aznar, Eduard Escalona (i2CAT)
Reviewers	Elio Salvadori (CNET), Michael Enrico (DANTE)

Abstract	This deliverable describes the software components for connectivity management, acting as network adapter or network orchestrator for the XIFI federation. Additionally, a brief description of the situation at M12 and short term evolution steps are provided for completeness.
Keywords	Network Controller, orchestrator, SDN

Document Revision History

Version	Date	Description of change	List of contributor(s)
V0.0	10/04/2014	Document creation	Luis M. Contreras (TID)
V0.1	13/04/2014	First release	Luis M. Contreras, Óscar González, Jaume Marhuenda (TID), José I. Aznar, Eduard Escalona (i2CAT)
V0.2	15/04/2014	Details about Trento test and Floodlight limitations	Giuseppe Cossu (CNET)
V1.1	22/04/2014	Details about installation instructions and system requirements, and NaaS content review	Jaume Marhuenda (TID), José I. Aznar (i2CAT)
V1.2	28/04/2014	Editorial review	Luis M. Contreras (TID)
V1.3	30/04/2014	Final release	Luis M. Contreras (TID)

Disclaimer

This report contains material which is the copyright of certain XIFI Consortium Parties and may only be reproduced or copied with permission in accordance with the XIFI consortium agreement.

All XIFI Consortium Parties have agreed to publication of this report, the content of which is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License[1].

Neither the XIFI Consortium Parties nor the European Union warrant that the information contained in the report is capable of use, or that use of the information is free from risk, and accept no liability for loss or damage suffered by any person using the information.

Copyright notice

© 2013 - 2015 XIFI Consortium Parties

Project co-funded by the European Commission in the 7th Framework Programme (2007-2013)		
Nature of the Deliverable:	P (Prototype)	
Dissemination Level		
PU	Public	✓
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to bodies determined by the Leone project	
CO	Confidential to Leone project and Commission Services	

[1] http://creativecommons.org/licenses/by-nc-nd/3.0/deed.en_US

EXECUTIVE SUMMARY

This document reports the components building the XIFI Network Controller. This controller is in charge of building the connectivity between the resources (virtual machines) deployed by the end user in the XIFI federation.

This controller orchestrates the needed connectivity devices to provide end-to-end connection.

The different components are reported, providing vision about basic functionality, component usage and installation guidelines.

Finally the status of the controller up to M12 is reported, including some details about short term evolution.

TABLE OF CONTENTS

EXECUTIVE SUMMARY **3**

TABLE OF CONTENTS **4**

LIST OF FIGURES **7**

LIST OF TABLES **8**

ABBREVIATIONS **9**

1 INTRODUCTION **10**

1.1 Scope 10

1.2 Document conventions 10

1.3 Intended audience 11

1.4 Reading suggestions 11

2 ABNO MODULE **12**

2.1 Summary 12

2.2 Component Responsible 13

2.3 Motivation 13

2.4 User Stories Backlog 13

2.5 State of the Art 14

2.6 Architecture Design 14

2.6.1 Basic actors 15

2.6.2 Main interactions 15

2.7 Release Plan 17

2.8 Installation Manual 17

2.8.1 Installation Requirements 17

2.8.2 Software Repository 18

2.8.3 Setup Guidelines 18

2.9 User Manual 20

2.9.1 API Specification 20

3 NAAS MODULE **24**

3.1 Summary 24

3.2 Component Responsible 24

3.3 Motivation 25

3.4 User Stories Backlog 25

3.5 State of the Art 26

3.6 Architecture Design 27
3.6.1 Basic actors 27
3.6.2 Main interactions 27
3.7 Release Plan 28
3.8 Installation Manual 28
3.8.1 Installation Requirements 28
3.8.2 Software Repository 28
3.8.3 Setup Guidelines 28
3.9 User Manual 29
3.9.1 API Specification 29
3.9.2 Handbook 31

4 XIFI NEUTRON PLUG-IN 34
4.1 Summary 34
4.2 Component Responsible 34
4.3 Motivation 35
4.4 User Stories Backlog 35
4.5 State of the Art 36
4.6 Architecture Design 36
4.6.1 Basic actors 36
4.6.2 Main interactions 36
4.7 Release Plan 36
4.8 Installation Manual 36
4.8.1 Installation Requirements 36
4.8.2 Software Repository 37
4.8.3 Setup Guidelines 37
4.9 User Manual 37

5 SITUATION AT M12 38
5.1 Network control tests performed 38
5.1.1 L2VPN connectivity test through the international network of Telefónica 38
5.1.2 Local point-to-point connectivity test in Trento node 39
5.2 Lite-NaaS component 41
5.3 Floodlight SDN controller 41

6 SHORT TERM EVOLUTION 42
6.1 Development of capabilities internal to the controller 42
6.1.1 OpenNaaS integration 42

6.1.2 QoS capabilities 42
6.1.3 New local SDN controller....43
6.2 Integration with elements external to the controller 44
6.2.1 Connection to MD-VPN transport infrastructure 44
6.2.2 Integration with the FI-Lab portal 45

REFERENCES 46

APPENDIX A: INTERACTION WITH FLOODLIGHT AS LOCAL SDN CONTROLLER 47

LIST OF FIGURES

Figure 1: XIFI Network Controller architecture 10
Figure 2: Building blocks considered for XIFI from the IETF ABNO framework 15
Figure 3: Workflow detailing ABNO components interactions.....16
Figure 4: Screenshot for point-to-point connectivity request.....21
Figure 5: Screenshot for point-to-point connectivity acknowledgment 21
Figure 6: Screenshot for broadcast connectivity request.....23
Figure 7: Screenshot for broadcast connectivity acknowledgement 23
Figure 8: OpenNaaS architecture 26
Figure 9: Connectivity service provisioning workflow.....27
Figure 10: Data centre view 32
Figure 11: User's VMs per data centre 32
Figure 12: Merge of the user's VMs 33
Figure 13: XIFI Network Controller test setup for L2VPN connectivity at Telefónica premises 38
Figure 14: XIFI Network Controller test at Trento node 40
Figure 15: Integration scenarios with MD-VPN 45

LIST OF TABLES

Table 1: ABNO module context details 12
Table 2: ABNO module dependencies summary 12
Table 3: Component responsible 13
Table 4: ABNO module user stories backlog 13
Table 5: Release plan for ABNO module 17
Table 6: System requirements for running the ABNO module 18
Table 7: NaaS module context details 24
Table 8: NaaS module dependencies summary 24
Table 9: Component responsible 25
Table 10: Component responsible 25
Table 11: NaaS component user stories backlog 26
Table 12: Release plan for ASO module (NaaS component) 28
Table 13: System requirements for running the ASO module (NaaS component) 28
Table 14: XIFI plug-in context details 34
Table 15: XIFI plug-in dependencies summary 34
Table 16: Component responsible 35
Table 17: Plug-in user stories backlog 35
Table 18: Release plan for XIFI plug-in 36
Table 19: System requirements for running the XIFI plug-in 37

ABBREVIATIONS

API	Application Program Interface
DC	Data Centre
GE	Generic Enabler
GRE	Generic Routing Encapsulation
VPN	Virtual Private Network
MAC	Media Access Control address
MD-VPN	Multi Domain Virtual Private Network
NREN	National Research and Education Network
L3-VPN	Layer 3 VPN
L2-VPN	Layer 2 VPN
PoC	Proof of Concept
QoS	Quality of Service
REST	Representational state transfer
UC	Use Case
VM	Virtual Machine
XIFI	eXperimental Infrastructures for the Future Internet

1 INTRODUCTION

1.1 Scope

This deliverable describes the XIFI Network Controller, including its main components and the API and interfaces existing among them.

The XIFI Network Controller plays that role of network orchestrator, and it has been already introduced in XIFI deliverable 3.1. Here it is briefly described for completeness.

The XIFI Network Controller has been designed combining both the ABNO (Application Based Network Orchestration) [1] and OpenNaaS [2] architectures. The following picture presents the modular architecture of the XIFI Network Controller including OpenNaas (green colour) and the specific components of the ABNO framework (red colour) considered for XIFI, since from the whole set of functional components proposed in ABNO, only some of them are needed.

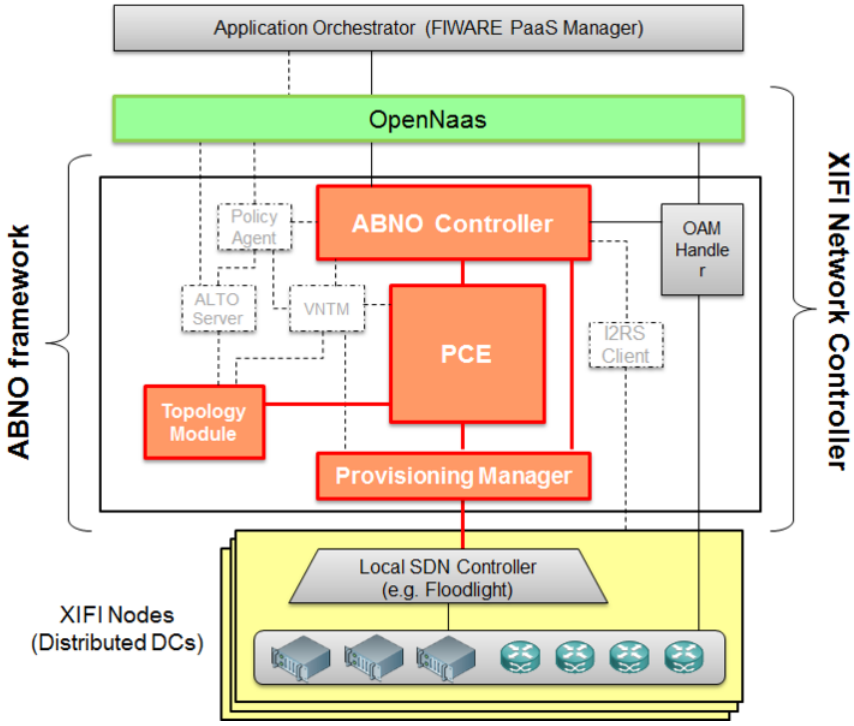


Figure 1: XIFI Network Controller architecture

In the XIFI Network Controller, the OpenNaaS framework provides the overall management of the connectivity service while keeping a logical view of the network thanks to the abstracted full-network topology orchestration provided by the ABNO part.

In the first stage of the project, a lite-NaaS module (with limited capabilities) called Application Service Orchestrator (ASO) has been developed and used instead of the target OpenNaaS component for testing the ABNO functionality. From the architecture point of view, the ASO module plays the same role as OpenNaaS.

1.2 Document conventions

The formatting of the document is compliant with the deliverable template provided by the XIFI

project. No other specific convention has been applied

1.3 Intended audience

The target audience of this deliverable is:

- The nodes of the XIFI federation (including original and new nodes)
- Experts and technical personnel providing deployment support and end-user support activities. These activities will be fulfilled by the support entity of the XIFI federation

1.4 Reading suggestions

The document is divided into six chapters. They report the status of the XIFI Network Controller and describe the APIs and the interfaces between the different software modules.

Chapter 1: Introduction

- Provides the information required to understand the rest of the document

Chapter 2: ABNO module

- Description of the ABNO component pertaining to the XIFI Network Controller

Chapter 3: NaaS module

- Description of the NaaS component used for the first public release of the XIFI Network Controller

Chapter 4: XIFI Neutron plug-in

- Description of the ad-hoc Neutron plug-in developed to be used with the XIFI Network Controller

Chapter 5: Situation at M12

- Reports the status at M12 of the XIFI Network Controller.

Chapter 6: Short term evolution

- Introduces the progress of the XIFI Network Controller in the short term for both the internal development of the controller and the external integration with other components of XIFI.

© 2013 - 2015 XIFI Consortium Parties

2 ABNO MODULE

2.1 Summary

This component composes the end-to-end connectivity by instructing local SDN controllers in each of the XIFI nodes to populate forwarding rules on the OpenFlow enable switches in a coordinated way, thus setting up the connectivity path (stitching the local connectivity to the transport capabilities in the WAN).

Reference Scenarios	UC-2 - Setup and usage of development environment UC-5 - Network and Data Centre operations
Reference Stakeholders	• Users: those who request a connected infrastructure comprising one or more XIFI nodes. • Infrastructure owners and operators: those who want to connect resources to the XIFI federation.
Type of Ownership	Development
Original tool	N/A
Planned OS license	GNU Affero General Public License
Reference OS Community	None at the moment

Table 1: ABNO module context details

Consists of	• ABNO Controller • Path Computation Element • Topology Module • Provisioning Manager
Depends on	• Local SDN Controller: for the first release Floodlight [2]open source controller has been used. Integration with Ryu[3] open source controller is on-going.

Table 2: ABNO module dependencies summary

[2] Floodlight Open SDN Controller: http://www.projectfloodlight.org/floodlight/

[3] Ryu SDN framework: http://osrg.github.io/ryu/

2.2 Component Responsible

Developer	Contact	Partner
Jaume Marhuenda	b.jmb@tid.es	TID
Óscar González	ogondio@tid.es	TID
Luis M. Contreras	lmcm@tid.es	TID

Table 3: Component responsible

2.3 Motivation

The XIFI project is a representative of an SDN controlled federation of distributed DCs. The delivery of distributed cloud services implies the configuration of various capabilities across the networks involved in the service (both in the distinct data centres and in the WAN domains connecting them). Within a data centre, the configuration of a cloud service is done using cloud management software to create virtual machines (VMs). Once the VMs have been created in each of the data centres and are connected internally to the DC, then a network connectivity service has to be invoked for connecting the overlay networks through the WAN.

The referred data centres represent separated administrative domains. Unfortunately the deployment of a single SDN controller in the federation to manage all the required connections does not scale. The existence of different administrative domains and a large number of devices complicates the operation and compromises the scalability of the controller. Nevertheless there is a requirement for a logically centralized entity maintaining a comprehensive view of all the network resources available in order to orchestrate them for providing the connectivity service. The ABNO component plays such role.

2.4 User Stories Backlog

Id	User Story Name	Actors	Description
1	Connect dispersed infrastructure	Infrastructure owners	The infrastructure owners provide infrastructure (computing and network) resources to the federation to interconnect them
2	Connect VMs	User	The user creates a VM in one or more XIFI nodes
3	Add new VMs to an existing service	User	The user deploys additional VMs in a different XIFI node
4	Total service removal	User	The user removes all the deployed VMs, and the connectivity resources are released

Table 4: ABNO module user stories backlog

2.5 State of the Art

As described before the ABNO component acts as orchestrator to coordinate separated and isolated SDN domains. The ABNO component has a centralized view of the topology and the available OpenFlow resources in each of the XIFI nodes (in terms of nodes, MAC addresses, ports in physical or virtual switches, etc).

At the time of conception of XIFI there was no other similar initiative. The ABNO framework has been defined in the IETF [1] for accomplishing such function, building on top of standard modules (as much as possible) communicated via standard interfaces (in some of the cases).

Nowadays there are some projects being developed in the same direction in terms of functionalities and capabilities. Specifically the OpenDayLight (ODL) project covers a very similar problem space.

ODL is an initiative promoted by the Linux Foundation. ODL is much more than a SDN controller since it incorporates many more functionalities which permit a more augmented control of the underlying network. ODL could be positioned in the future as the generic orchestrator of a connectivity service playing a similar role to the ABNO nowadays. Some of the components of the ODL architecture can be mapped (maybe not totally) to the ABNO building blocks. By now ODL is not yet sufficiently mature to substitute the ABNO component, but one of the activities in progress in T3.1 is the monitoring of ODL as a community development alternative for the future.

2.6 Architecture Design

The ABNO component follows the ABNO framework being defined by the IETF for an efficient provision of services according to application requests.

The IETF ABNO architecture is based on existing standard blocks defined within the IETF (PCE, ALTO, VNTM...), which could be implemented either in a centralized or distributed way according to network operator requirements. Thanks to the modular nature of the ABNO architecture, building blocks could be deployed by different developers or third parties and even by a single developer. This modularity and the standard interfaces between the modules could allow functional evolution of the complete architecture. Additionally, ABNO is especially well adapted to multi-domain scenarios by interacting with different SDN controllers just providing the required adapter in the South bound interface.

Not all the modules proposed in the ABNO framework are used for XIFI. The following figure shows a schematic view of the modules considered for this project.

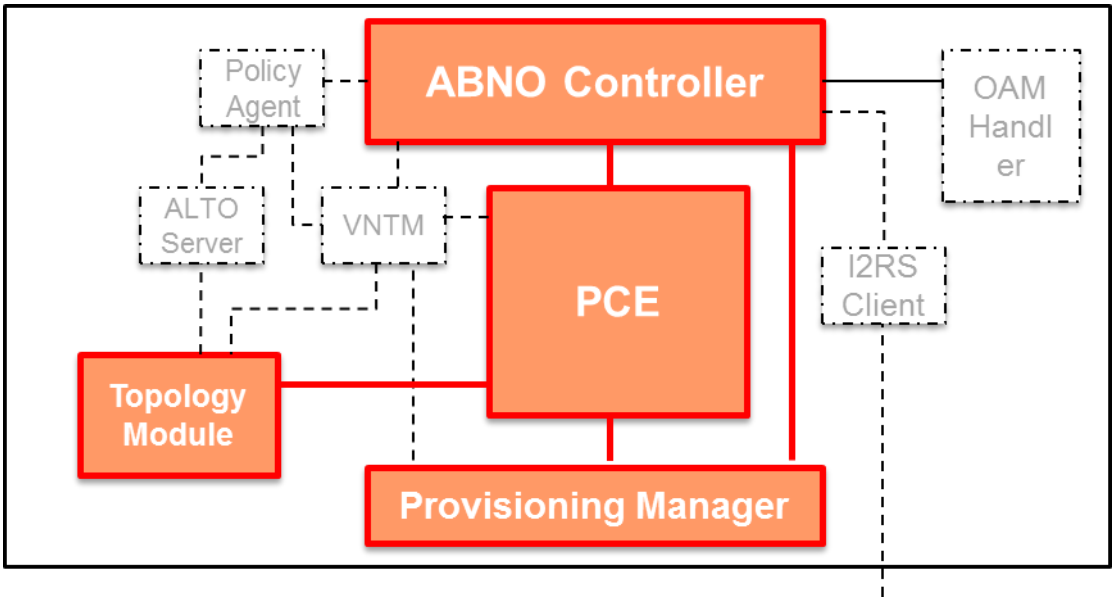


Figure 2: Building blocks considered for XIFI from the IETF ABNO framework

2.6.1 Basic actors

The ABNO component consists of a number of modules:

- The ABNO Controller stores a repository of workflows for operations in the network (e.g., connectivity provisioning). It is the main component of the architecture and is responsible for orchestrating, and invokes the necessary components in the right order. It listens for requests from the North bound interface and selects the appropriate workflow to follow in order to satisfy each request.
- The Path Computation Element (PCE) is the unit that handles the path computation across the network graph. It can calculate traffic engineered end-to-end paths.
- The Topology Module (TM) handles databases with topology information for allowing advanced services like traffic engineering.
- Finally, the Provisioning Manager (PM) is the module in charge of configuring the network elements by configuring the resources in each node. To do that, the PM will use the APIs of the local SDN controllers deployed in each node being a separate SDN domain.

2.6.2 Main interactions

The following diagram presents the interactions between the actors of the ABNO component. Note that the referenced ASO component (as temporal solution before integrating with the targeted OpenNaaS component) is explained later. For the purpose of this work, the ASO component has to be considered as the source triggering the configuration actions in the ABNO part.

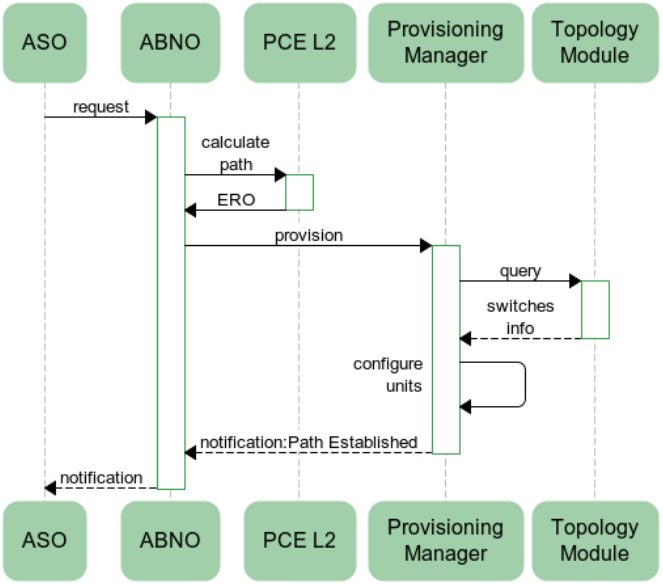


Figure 3: Workflow detailing ABNO components interactions

The following steps are required to provision a point-to-point link using the ABNO:

0. The initial trigger for this use case is a request from an end user via, typically, a portal. In this case ASO will do this request.

1. The ASO sends the required information to the ABNO controller including the two switch identifiers, the source and destination MACs and port through which the hosts are attached to the switch.

    a. This interface to the ABNO controller is, for example, assumed to be an HTTP/JSON interface.

2. When the ABNO controller receives the request, it asks the PCE for a path between the two hosts.

    a. PCEP is used in this interface.

    b. Some extensions are required to the current protocol definitions so some information such as the switch identifiers which are made of 10 bytes are coded into newly defined ERO sub-objects.

3. The PCE replies to the ABNO controller with a PCEP response.

    a. The PCEP protocol is extended in order to fill the PCEP response.

4. The ABNO controller creates a PCEP initiate packet from the response and sends it to the Provision Manager (PM).

    a. This step is done using PCEP.

    b. The ERO inside the Initiate is the same as the one inside the response so the PCEP protocol is extended in the same way as in the previous step.

5. The Provisioning Manager queries the Topology Module for the description of each node.

6. Depending on the technology and configuration mode selects a different protocol to complete the request. OF controller is queried.
    a. This time technology will always be OF but the configuration mode can vary from node to node
7. OF controller configures OF switches.
8. Once the path has been established it notifies the ABNO controller.
    a. This interface is PCEP.
9. Similarly, the ABNO controller advertises the ASO.
    a. This interface is JSON.

The mentioned OF controller refers to the local SDN controller deployed in the XIFI node.

2.7 Release Plan

Version ID	Features	Milestone	User Stories
v1.0	• First release of ABNO modules • Workflow for L2 connectivity • South Bound Interface with Floodlight as local SDN controller	M7	1, 2
v1.1	• Support of Ryu as local SDN controller	M12	1, 2
v2.0	• Integration with OpenNaaS	M13	3, 4

Table 5: Release plan for ABNO module

2.8 Installation Manual

2.8.1 Installation Requirements

The following table summarizes the system requirements for healthy running the XIFI Network Controller.

Operating system	Ubuntu 12.04 or higher
Memory	4 GB RAM
Hard drive	80 GB available in the hard disk
Java	Version 1.7.0_45 or higher

Table 6: System requirements for running the ABNO module

2.8.2 Software Repository

For the time being, the software will be delivered via a Dropbox folder. It is foreseen to host the software in the project SVN and in a GIT repository once the software is available as open source software.

The public Dropbox link where the component can be found is the following:

https://www.dropbox.com/s/d78kgo89mtnmj9o/TID_xifi_package_v1.0.zip

https://www.dropbox.com/s/8t5rnl0ss28t2n1/TID_xifi_package_v1.1.zip

2.8.3 Setup Guidelines

This documentation assumes the file README.md inside the root of the TID installation package has also been read.

Install Java JRE environment:

```
http://www.liberiangeek.net/2012/04/install-oracle-java-runtime-jre-7-in-ubuntu-12-04-precise-pangolin/
```

Install Devstack Grizzly (https://github.com/openstack-dev/devstack/archive/stable/grizzly.zip) and configure the localrc like this:

```
#HOST_IP=172.16.104.202
#FLAT_INTERFACE=eth1
#MULTI_HOST=1
LOGFILE=/opt/stack/logs/stack.sh.log

disable_service n-net
enable_service q-svc
enable_service q-dhcp
enable_service neutron
enable_service bigswitch_floodlight
Q_PLUGIN=bigswitch_floodlight
Q_USE_NAMESPACE=False
NOVA_USE_NEUTRON_API=v2
```

```
MYSQL_PASSWORD=tid
RABBIT_PASSWORD=tid
ADMIN_PASSWORD=tid
SERVICE_PASSWORD=tid
SERVICE_TOKEN=tokentoken
DEST=/opt/stack
SCREEN_LOGDIR=$DEST/logs/screen

SYSLOG=True

SCHEDULER=nova.scheduler.simple.AggregateInstanceExtraSpecsFilter,nova.sche
duler.simple.AvailabilityZoneFilter,nova.scheduler.simple.RamFilter,nova.sc
heduler.simple.ComputeFilter

#IP:Port for the BSN controller
#if more than one, separate with commas
BS_FL_CONTROLLERS_PORT=10.95.164.238:6633
BS_FL_CONTROLLER_TIMEOUT=10
Q_USE_NAMESPACE=True
```

The password should be replaced with the one you choose.

Run these commands:

```
./unstack.sh
./stack.sh
```

Run Floodlight after configuring `init_setup_2.sh`, e.g.,

```
STACK_DIR=devstack-stable-grizzly
PLUGIN_DIR=TID_plugin
ABNO_DIR=ABNO
ASO_DIR=pip
PORT=eth0
LOG_FILE=setup.log
KILL_TAG=quantum-server
```

You should generally only need to change the PORT variable. This is the interface through which the physical host will be connected to other Openstack instances, not the management IP of the physical host.

```
./init_setup_2.sh -Floodlight
```

And check if it is running and the IP of the controller with

```
ps aux | grep floodlight
sudo ovs-vsctl get-controller br-int
```

If it is not the IP that you expected (e.g. localhost) run:

```
sudo ovs-vsctl set-controller br-int  tcp:127.0.0.1:663
```

Replace the plugin with:

```
./init_setup_2.sh -Plugin
```

The plugin has a configuration file specified in README.md which should be properly filled

After that, you can run the ABNO:

```
./init_setup_2.sh -ABNO
```

The ABNO has several configuration files specified in README.md

2.9 User Manual

2.9.1 API Specification

Create unicast path

Verb	URI
GET	`source=<?>&destination=<?>&source_mac=<?>&dest_mac=<?>&source_interface=<?>&destination_interface=<?>&operation_type=<?>&id_operation'`

Action:

Creates a unicast unidirectional link between two hosts identified by their MAC addresses..

Parameter Description:

- Operation_Type (mandatory). This string is mapped to the workflow that the ABNO controller has to execute. For this workflow the value is: “MPLSProvisioningWF”.
- Source and Destination Switches (mandatory). These variables contain the switch identifiers given by floodlight.
- Source and destination MACs of the VMs (mandatory): the MACs from the VMs that are the origin and the destination of the LSP.
- Source interface is the connection port of the source VM to the source Switch (mandatory): the physical port that connects the source VM to the source Switch. This is necessary to correctly create the OpenFlow rules.
- Destination interface is the connection port of the destination VM to the destination Switch (mandatory): the physical port that connects the destination VM to the destination Switch.
- ID_Operation is the identifier for the specific operation in order to trace back and reference this specific action in the network.

Response:

Returns a JSON containing the configuration parameters.

Normal Response Code: 200

Error Response Code: any other.

Example:

A request could be done in the following way:

```
curl
'localhost:4445?source=10:00:2c:59:e5:66:ed:00&destination=10:00:2c:5
9:e5:64:21:00&source_mac=00:1e:c9:bb:7e:54&dest_mac=00:13:3b:52:02:27
&source_interface=20&destination_interface=20&Operation_Type=
BandwidthProvisioningWF &ID_Operation=1234'
```

And the response could be:

```
{
    "ID_Operation":"15697"
    "Operation Type":"BANDWIDTH_PROVISIONING",
    "Result": "LINK_CONFIGURED"
    "Error Code": "NO_ERROR"
}
```

The following figure shows a query to the ABNO asking for this workflow to be executed.

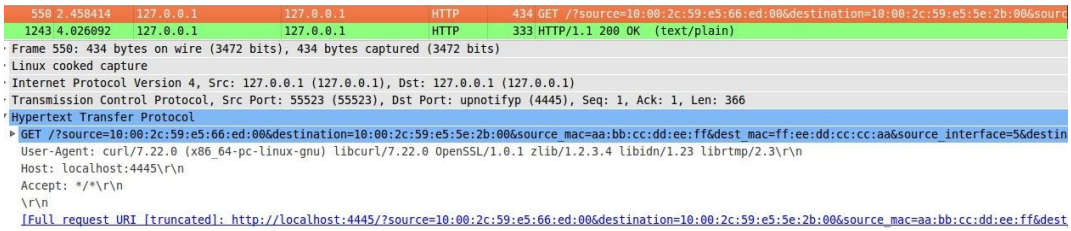


Figure 4: Screenshot for point-to-point connectivity request

Additionally, the following capture shows a response from the ABNO where the point to point path has been configured correctly.

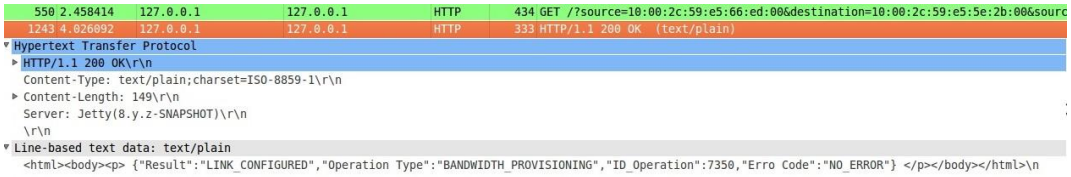


Figure 5: Screenshot for point-to-point connectivity acknowledgment

Create broadcast tree

Verb	URI	DATA
POST	Operation_Type=BroadcastProvisioningWF&ID_Operation=1234	JSON containing the necessary parameter to build the tree. More on this in the example.

Action:

Creates a broadcast tree. After the workflow is run if a packet is sent with MAC destination "`ff:ff:ff:ff:ff:ff`" from the source node contained in data, this packet arrives to all the endpoints contained in data. Notice that to create a broadcast domain this workflow has to be run once for each node in the net with endpoint the rest of the nodes.

Parameter Description:

- Operation_Type (mandatory). This string is mapped to the workflow that the ABNO controller has to execute. For this workflow the value is: "BroadcastProvisioningWF".
- ID_Operation is the identifier for the specific operation in order to trace back and reference this specific action in the network.

Parameters encapsulated in DATA

The DATA part of the POST request must contain a structure similar to this:

- Source Switch (mandatory). This variable contains the switch identifier given by floodlight.
- Source MACs of the VMs (mandatory): the MAC from the VM that is the origin and the destination of the LSP.
- List of destination VMs (mandatory): this list will contain all the VMs that will receive. Every item in this list will contain the switch to which the VM is attached and the MAC of the VM

```
{
	"data": [
	{
		"port": "2",
		"switch": "00:00:00:1e:c9:bb:7e:54"
	},
	{
		"port": "3",
		"switch": "00:00:00:1e:c9:bb:7e:54"
	},
	{
		"port": "15",
		"switch": "00:00:00:1c:c4:da:ba:c2"
	},
	{
		"port": "16",
		"switch": "00:00:00:1c:c4:da:ba:c2"
	}
	],
	"source_port" : 2
	"source_mac": "FA:16:3E:06:C4:B2",
	"source_switch_id": "00:00:00:1e:c9:bb:7e:54"
}
```

The data variable contains the endpoint at which the broadcast message will arrive. The source_port, souce_mac and source_switch_id identify the origin.

Example:

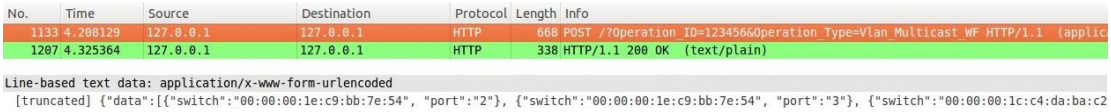
Request:

```
curl -d '{"data": [{"port": "2","switch":
"00:00:00:1e:c9:bb:7e:54"},{"port": "3","switch":
"00:00:00:1e:c9:bb:7e:54"},{"port": "15","switch":
"00:00:00:1c:c4:da:ba:c2"},{  "port": "16","switch":
"00:00:00:1c:c4:da:ba:c2"}],"source_port" : 2"source_mac":
"FA:16:3E:06:C4:B2","source_switch_id": "00:00:00:1e:c9:bb:7e:54"}'
'localhost:4445?source_switch_id=10:00:2c:59:e5:66:ed:00&source_mac=FA:
16:3E:06:C4:B2&source_port=2&Operation_Type=Vlan_Multicast_WF&ID_Operat
ion=1234'
```

Response:

```
{
    "ID_Operation":"15697"
    "Operation Type":"MULTICAST_PROVISIONING",
    "Result": "LINK_CONFIGURED"
    "Error Code": "NO_ERROR"
}
```

The following figure shows a query to the ABNO asking for this workflow to be executed.

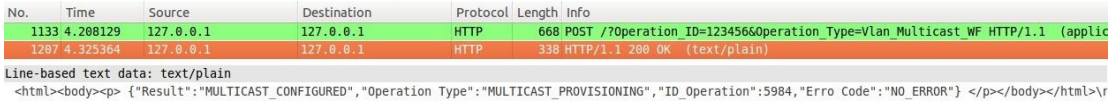


No.	Time	Source	Destination	Protocol	Length	Info
1133	4.208129	127.0.0.1	127.0.0.1	HTTP	668	POST /?Operation_ID=123456&Operation_Type=Vlan_Multicast_WF HTTP/1.1 (applic
1207	4.325364	127.0.0.1	127.0.0.1	HTTP	338	HTTP/1.1 200 OK (text/plain)

Line-based text data: application/x-www-form-urlencoded
[truncated] {"data":[{"switch":"00:00:00:1e:c9:bb:7e:54", "port":"2"}, {"switch":"00:00:00:1e:c9:bb:7e:54", "port":"3"}, {"switch":"00:00:00:1c:c4:da:ba:c2

Figure 6: Screenshot for broadcast connectivity request

Additionally, the following capture shows a response from the ABNO where the path has been configured correctly.



No.	Time	Source	Destination	Protocol	Length	Info
1133	4.208129	127.0.0.1	127.0.0.1	HTTP	668	POST /?Operation_ID=123456&Operation_Type=Vlan_Multicast_WF HTTP/1.1 (applic
1207	4.325364	127.0.0.1	127.0.0.1	HTTP	338	HTTP/1.1 200 OK (text/plain)

Line-based text data: text/plain
<html><body> {"Result":"MULTICAST_CONFIGURED","Operation Type":"MULTICAST_PROVISIONING","ID_Operation":5984,"Erro Code":"NO_ERROR"} </body></html>\r

Figure 7: Screenshot for broadcast connectivity acknowledgement

3 NAAS MODULE

3.1 Summary

The NaaS module is required in the XIFI Network Controller to provide connectivity service awareness across the federation. It is in charge of collecting and maintaining the information about the resources committed to the end user from the connectivity point of view, either if the resources are local to just one XIFI node or if they are spread in different nodes in the federation.

At the first stage of the XIFI Network Controller development the module provided is a temporal one for ABNO testing. It has been called Application Service Orchestrator (ASO) and it is used to maintain the notion of connectivity service end to end by associating VMs located in separated data centres with the required network connection among them.

AS described later in this deliverable forthcoming XIFI Controller releases will substitute this component with the service awareness functionality/module of the OpenNaaS platform [2-4].

Reference Scenarios	UC-2 - Setup and usage of development environment UC-5 - Network and Data Centre operations
Reference Stakeholders	• Users: those who request a connected infrastructure comprising one or more XIFI nodes. • Infrastructure owners and operators: those who want to connect resources to the XIFI federation.
Type of Ownership	Development
Original tool	N/A
Planned OS license	GNU Affero General Public License
Reference OS Community	None at the moment

Table 7: NaaS module context details

Consists of	• Application Service Orchestrator
Depends on	• ABNO module

Table 8: NaaS module dependencies summary

3.2 Component Responsible

The current status of the XIFI Network Controller will first include the ASO component for ABNO testing purposes. The ASO component will be replaced by the OpenNaaS component to provide to the XIFI Network Controller with the full required features.

Respect to the ASO component for ABNO test harness, the responsible team is:

Developer	Contact	Partner
Jaume Marhuenda	b.jmb@tid.es	TID
Óscar González	ogondio@tid.es	TID
Luis M. Contreras	lmcm@tid.es	TID

Table 9: Component responsible

Respect to the future-proof OpenNaaS component for complementing the XIFI Network Controller, the responsible team is:

Developer	Contact	Partner
José I. Aznar	jose.aznar@i2cat.net	i2CAT
Eduard Escalona	eduard.escalona@i2cat.net	i2CAT

Table 10: Component responsible

Currently both teams are working together to define the OpenNaaS functionality in XIFI and to ensure the smooth transition from actual release to future releases.

3.3 Motivation

Once an end user request resources in the form of VMs from the federation it is required to provide connectivity among those resources. Such a service can also changes in time by either modifying the resources involved, or even by completely deleting the previously committed resources.

Any of the above operations require to store the information of the service provided to the end user. This service awareness is kept and handled by the NaaS module (in the ASO component at this time of the project, OpenNaaS in short term).

3.4 User Stories Backlog

Id	User Story Name	Actors	Description
1	Connect dispersed infrastructure	Infrastructure owners	The infrastructure owners provide infrastructure (computing and network) resources to the federation to interconnect them
2	Connect VMs	User	The user creates a VM in one or more XIFI nodes
3	Add new VMs to an existing service	User	The user deploys additional VMs in a different XIFI node
4	Total service removal	User	The user removes all the deployed VMs, and the connectivity resources

Id	User Story Name	Actors	Description
			are released

Table 11: NaaS component user stories backlog

3.5 State of the Art

One of the functionalities of the NaaS framework enables to abstract the underlying network complexity allowing the possibility for easily deploy and operate customized advanced network services according to end user requests. These capabilities allow the extension the IaaS model from the Datacentres to access and transport networks.

The NaaS model has been brought forward with the OpenNaaS platform for easy prototyping and proof casing of XIFI network connectivity service. OpenNaaS is an open-source framework, developed under the FP7 MANTYCHORE project, which provides (among others) tools for managing the different resources present in any network infrastructure.

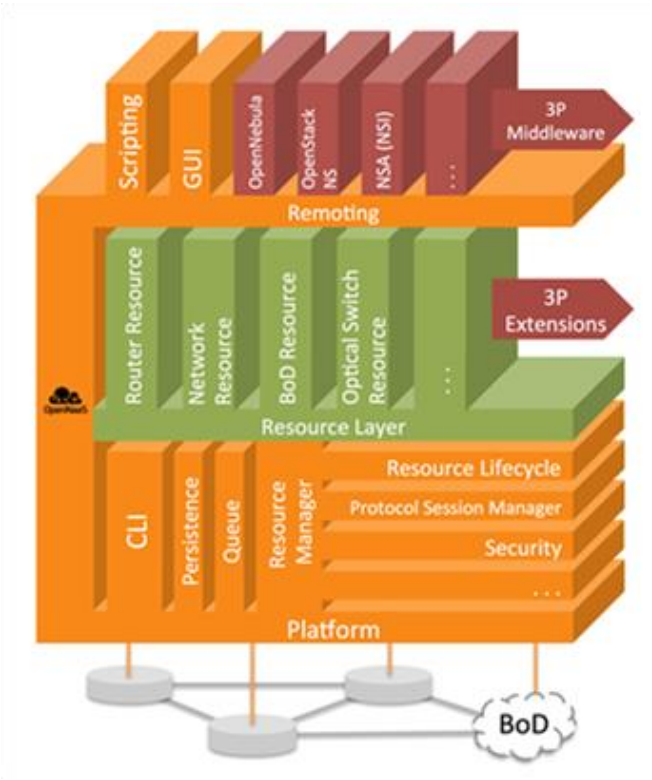


Figure 8: OpenNaaS architecture

Inside XIFI the OpenNaaS framework is part of the network controller architecture (see [5] for more complete description of the component), enabling users to access the isolated control and management of the infrastructure provided to connect the XIFI nodes.

OpenNaaS will provide more advanced NaaS capabilities to complement the XIFI Network Controller. Some of these capabilities are:

- Specific databases for storing the resources allocated per tenant, including location (XIFI node where these resources are deployed).

- Association (and possibly, allocation) of the identifiers involved in the service, like network identifier in the OpenStack instances for a given tenant, connectivity service identifier, tenant identifier, etc.
- Support of multiple networks per tenant, if needed.

For the current release, the ASO module presents basic NaaS capabilities for testing the ABNO part.

3.6 Architecture Design

The Application Service Orchestrator (ASO) is a component used to maintain the number of networks created (limited to one per tenant or end user in this case) created and their interconnection with the physical end points of the network. Moreover, ASO contains information about the virtual and physical switches in each data centre.

3.6.1 Basic actors

The ASO is the sole actor.

The configuration of the OpenFlow-enabled switches (either physical or virtual) must be triggered through the ASO, which will maintain the number of networks created and their corresponding interconnection with the physical end points of the network.

Once the VMs are attached to the switched infrastructure it is then possible to create a network containing VMs from multiple data centres. The ABNO is called from the ASO to create a multisite point-to-point connection.

The interaction between ASO and ABNO was graphically shown in the previous chapter.

3.6.2 Main interactions

From the portal a number of VMs will be created in the Openstack instance (step 0). Openstack will inform the Application Service Orchestrator (ASO) (step 1) and ASO will process the request and configure the Openstack if necessary (step 2 and 3).

After these steps, it is assumed there are different networks in different instances of Openstack. In order to join some of these networks the portal will directly query the ASO (step 4) and the ASO will configure the Openstack instances to join the networks (steps 5, 6, 7).

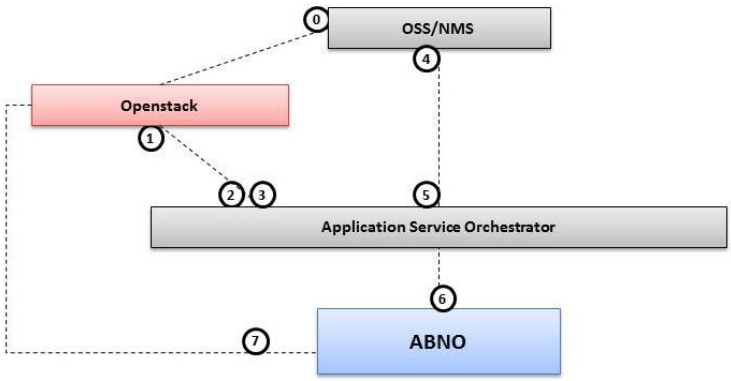


Figure 9: Connectivity service provisioning workflow

3.7 Release Plan

Version ID	Features	Milestone	User Stories
v1.0	• First release for emulating NaaS / Portal capabilities for acting as test harness of the ABNO component	M7	1, 2

Table 12: Release plan for ASO module (NaaS component)

No more releases of ASO are foreseen. The ASO component will be deprecated once the integration between ABNO and OpenNaaS is finished. The user stories 3 and 4 will be covered with OpenNaaS.

3.8 Installation Manual

3.8.1 Installation Requirements

The following table summarizes the system requirements for healthy running the NaaS component provided by the ASO.

Operating system	Ubuntu 12.04 or higher
Memory	4 GB RAM
Hard drive	80 GB available in the hard disk
Python	Version 2.7

Table 13: System requirements for running the ASO module (NaaS component)

3.8.2 Software Repository

For the time being, the software will be delivered via a Dropbox folder. It is foreseen to host the software in the project SVN and in a GIT repository once the software is available as open source software.

The public Dropbox link where the component can be found is the following:

https://www.dropbox.com/s/d78kgo89mtnmj9o/TID_xifi_package_v1.0.zip

3.8.3 Setup Guidelines

After having setup the ABNO component, at the root path of the installation package execute the following instruction:

```
./init_setup_2.sh -ASO
```

Before doing so the configuration file of the ASO must be modified. The file ./plugin_config_file.xml must be completed with the IP of the ABNO that will control the instance, where ./ is the root of the installation folder

3.9 User Manual

3.9.1 API Specification

1) Called from Openstack when a host is added to a net.

URL	/add_host_to_network
Parameters	The whole lot of parameters can be seen in the example. The important ones are the tenant_id, the id, the mac_address, the network_id to which the VM will be attached, the IP address and the device_owner. The port number can be obtained from the id of the machine and the switch information provided by floodlight. Below there are two examples. They are interesting because the first ones correspond to a DHCP server and the second one to a VM. The way to differentiate them is by the device_owner variable.
Example	`curl -d '{'name': '', 'network_id': u'6188dd39-c04d-46d1-8735-7c80f94dc4dd', 'tenant_id': u'9910f839027041979600db22d178131b', 'state': 'DOWN', 'device_owner': 'network:dhcp', 'mac_address': 'fa:16:3e:67:a4:5a', 'fixed_ips': [{'subnet_id': u'178c8566-0170-40b7-990b-47deacb0ce11', 'ip_address': u'10.0.4.1'}], 'id': 'a5e3c410-4261-49c4-b970-9ca326a4db75', 'device_id': u'dhcp7cb3c7e0-4768-57c7-aa46-768d1215deb8-6188dd39-c04d-46d1-8735-7c80f94dc4dd'}' http://10.95.164.243:5002/add_host_to_network`  `curl -d '{'name': '', 'network_id': u'6188dd39-c04d-46d1-8735-7c80f94dc4dd', 'tenant_id': u'9910f839027041979600db22d178131b', 'state': 'DOWN', 'device_owner': u'compute:None', 'mac_address': 'fa:16:3e:7a:1c:e8', 'fixed_ips': [{'subnet_id': u'178c8566-0170-40b7-990b-47deacb0ce11', 'ip_address': u'10.0.4.5'}], 'id': 'd2b4db65-2745-40a4-b124-3755661d28fb', 'device_id': u'48da11ab-ee13-45bc-ac9c-dea2c57bcb70'}' http://10.95.164.243:5002/add_host_to_network`

It always assumes that VMs are attached to the OpenVSwitch and configures the OpenVSwitch to create a net.

2) Called from Openstack when a net is created.

URL	/create_network
Parameters	The parameters can be viewed in the example below. The important ones are the id and the tenant_id. The net will be referenced in the future by this id.
Example	`curl -d '{'status': 'ACTIVE', 'subnets': [], 'name': u'net1', 'admin_state_up': True, 'tenant_id': u'9910f839027041979600db22d178131b', 'router:external': False, 'shared': False, 'id': '6188dd39-c04d-46d1-8735-7c80f94dc4dd'}' http://10.95.164.243:5002/create_network`

This doesn't configure any rule in the OpenVSwitch.

3) This is used to print the topology from the ASO Layer.

URL	/print_topology
Parameters	None
Example	curl localhost:5002/print_topology

Output	{
	"433c1c1e-4f59-4cfc-a359-5f037e7fef92": {
	"00:00:00:1c:c4:d8:35:02": {
	"floodlight": "localhost",
	"ports_macs": [
	{
	"ip": "unkown",
	"mac": "fa:16:3e:f0:6b:7e",
	"port": 33
	}
	],
	"priority": 1010
	},
	"00:00:00:1e:c9:bb:7e:54": {
	"floodlight": "localhost",
	"ports_macs": [
	{
	"ip": "unkown",
	"mac": "fa:16:3e:8d:67:d7",
	"port": 89
	}
	],
	"priority": 1010

	<pre>}
	}
	}</pre>

It assumes OpenStack is responsible of attaching the VM to the Open vSwitch and it configures the OpenVSwitch to create a net.

4) This is used to initialize the ASO database without having to create VM machines thought Openstack.

URL	/read_topology
Parameters	All the network info
Example	<pre>curl -d '{"1cd36c6b-7d66-4101-a3d6-0574345ca03c": {"10:00:2c:59:e5:66:ed:00": {"priority": 1010, "ports_macs": [{"mac": "00:1e:c9:bb:7e:54", "port": 19}], "floodlight": "10.95.164.243"}}, "1cd36c6b-7d66-4101-a3d6-0574345ca032": {"10:00:2c:59:e5:5e:2b:00": {"priority": 1010, "ports_macs": [{"mac": "00:1c:c4:d8:35:02", "port": 19}], "floodlight": "10.95.164.244"}}}' 10.95.164.243:5002/read_topology</pre>

5) Called in order to join to networks

URL	/join_networks
Parameters	The two identifiers of the net

Example	<pre>curl -d '{"net_id_1":645d5ded-08cc-46da-a3d0-fe797a6af7a4
	, "net_id_2":8ceda6c6-9009-48ec-b2e5-ed56a45372d2
	}'http://10.95.164.243:5002/join_networks</pre>

As a result of this call, the rules in the switches are changed, including unicast and multicast rules.

3.9.2 Handbook

The following steps are executed to provide end-to-end connectivity:

1. The VMs are connected through a virtual switch called OpenVSwitch. Some OpenFlow rules such as dropping any unknown packets are created in them. All the ABNO modules are started (ABNO Controller, Provisioning Manager, Topology Module and PCE) and the ASO.
2. Create two nets, one in each data-centre. This will create two DHCP servers. This step calls the ASO API REST function /create_network, previously described. Therefore, no rules are created in the OpenFlow switches.

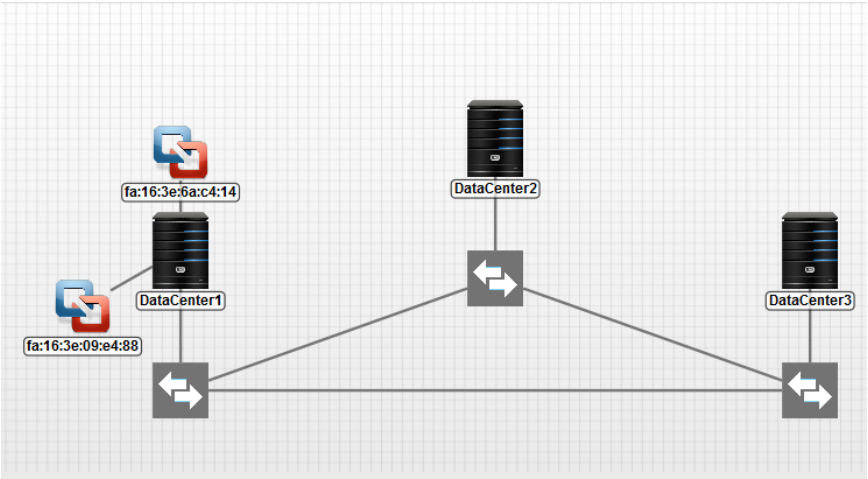


Figure 10: Data centre view

3. Create two VMs, and attach each one to each net. The result will be that each net now has two VMs, the DCHP server and the normal VM. This time rules will be created in the OpenFlow switches so that it truly becomes a net. In order to accomplish this broadcast rules and an LSP between the VM and the DHCP server will be created.

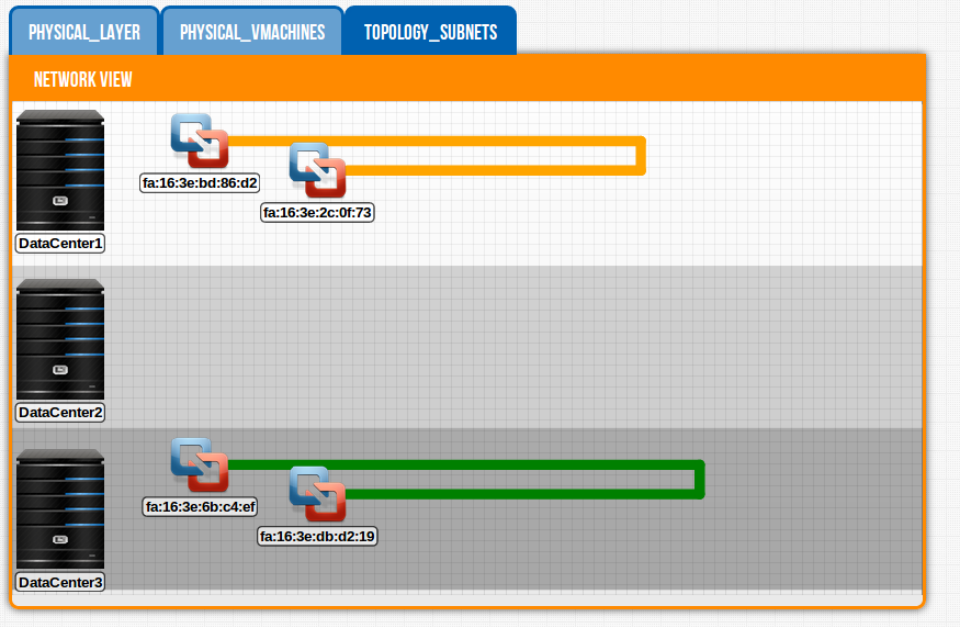


Figure 11: User's VMs per data centre

4. Note that no there is not yet any connectivity between the datacentres. Now ASO will call ABNO and ask to join the two networks, configuring the HP switches and the OV switches, thus giving L2 connectivity. The result after the union can be viewed in the frontend and should be something similar to this:

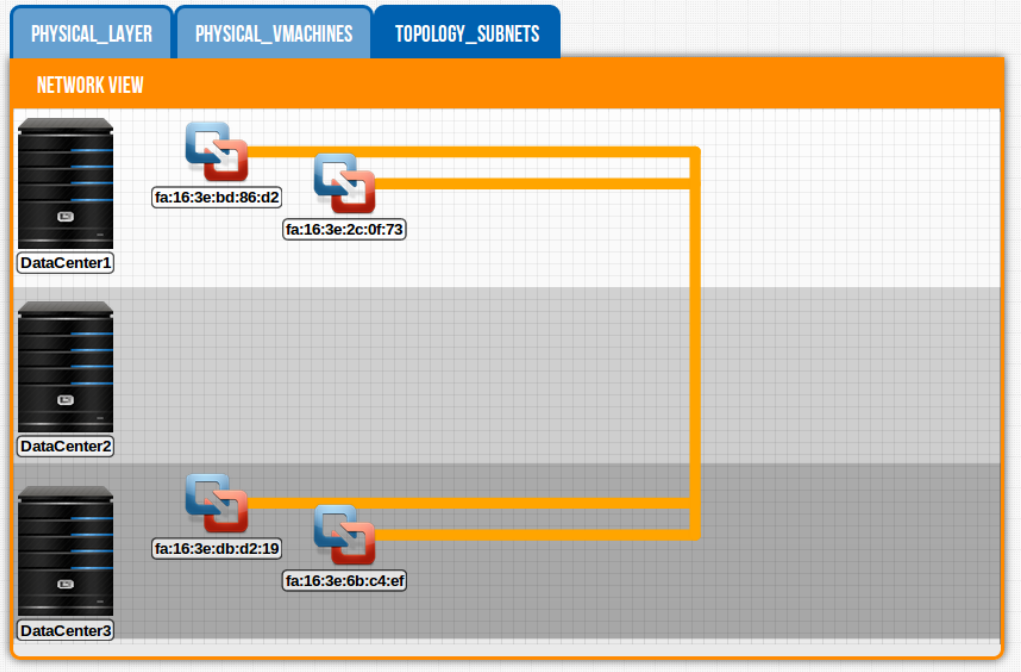


Figure 12: Merge of the user's VMs

4 XIFI NEUTRON PLUG-IN

4.1 Summary

OpenStack's Neutron component is in charge of providing the local overlay networks. It works with a technology dependent plugin. For XIFI project a customized plug-in has been developed to manage the internal datacentre (DC) connectivity. The plug-in interacts with the XIFI Network Controller, which configures the Open vSwitch[4] of each compute node of the OpenStack instance running in each of the XIFI nodes. The XIFI Neutron plug-in is released as Open Source in the framework of the XIFI project.

Reference Scenarios	UC-2 - Setup and usage of development environment UC-5 - Network and Data Centre operations
Reference Stakeholders	• Users: those who request a connected infrastructure comprising one or more XIFI nodes. • Infrastructure owners and operators: those who want to connect resources to the XIFI federation.
Type of Ownership	Development and extension
Original tool	OpenStack
Planned OS license	GNU Affero General Public License
Reference OS Community	None at the moment

Table 14: XIFI plug-in context details

Consists of	• Application Service Orchestrator
Depends on	• ABNO module

Table 15: XIFI plug-in dependencies summary

4.2 Component Responsible

Developer	Contact	Partner
Jaume Marhuenda	b.jmb@tid.es	TID
Óscar González	ogondio@tid.es	TID

[4] Open Virtual Switch, Open vSwitch: http://openvswitch.org/

Luis M. Contreras	lmcm@tid.es	TID

Table 16: Component responsible

4.3 Motivation

The Neutron component in OpenStack is in charge of setting up the connectivity between the VMs and the OpenFlow infrastructure in the DC. Neutron requires a technology specific plug-in to directly interact with a local SDN controller to request the appropriate flow forwarding entries.

The use case identified inside XIFI requires the control of the connectivity of a multi-domain environment. Thus the installation of the forwarding entries inside the switches has to be coordinated by the XIFI network controller. Additionally, the number of rules installed into the switches in has to be be strictly controlled for security reasons. This is because the XIFI nodes are parts of production environments.

For this reason a specific XIFI Neutron plug-in has been configured from the original Floodlight plug-in collaboration with OpenStack. The basic differences between the XIFI and the original Floodlight plug-in are the following:

- No forwarding rules are installed at the beginning (e.g., for DHCP). The intention is to ensure a complete proactive behavior from the SDN point of view. The initial situation is not allowing any traffic. Then, once a connectivity service is required, the rules strictly necessary (both for unicast and broadcast traffic) will be created (including connectivity to the DHCP server).
- Conventional Neutron plug-in directly interacts with the local SDN controller for configuring rules in the switching devices. On the contrary, the XIFI plug-in does not interact with the SDN controller local to the DC in a direct manner, but it interacts with the XIFI Network Controller through the NaaS component, which in turn triggers actions on the OpenFlow infrastructure deployed in the DC. This ensures a total control of the rules pushed to the forwarding elements from a federation-wide connectivity perspective.

4.4 User Stories Backlog

Id	User Story Name	Actors	Description
1	Connect dispersed infrastructure	Infrastructure owners	The infrastructure owners provide infrastructure (computing and network) resources to the federation to interconnect them
2	Connect VMs	User	The user creates a VM in one or more XIFI nodes
3	Add new VMs to an existing service	User	The user deploys additional VMs in a different XIFI node
4	Total service removal	User	The user removes all the deployed VMs, and the connectivity resources are released

Table 17: Plug-in user stories backlog

4.5 State of the Art

OpenStack Grizzly is the software chosen for handling the computing resources. There is an independent OpenStack instance per data centre. All the services (Nova, Neutron, Cinder, Glance and Keystone) run in the same physical machine. According to OpenStack terminology, a server which is able to run virtual machines is called a "compute node".

4.6 Architecture Design

The XIFI plug-in does not present differences from the architecture point of view with respect to the conventional OpenStack Neutron architecture. It just adapts the already existing Floodlight Neutron plug-in. In this way, the Neutron component can be integrated in the workflows of the XIFI Network Controller for federation control.

4.6.1 Basic actors

The XIFI Neutron plug-in is the sole actor.

4.6.2 Main interactions

4.7 Release Plan

Version ID	Features	Milestone	User Stories
v1.0	• First release for OpenStack Grizzly	M7	1, 2, 3, 4

Table 18: Release plan for XIFI plug-in

No more releases of the XIFI Neutron plug-in for Grizzly distribution are foreseen.

New releases will be required according to the OpenStack releases. The next one is Havana, and in consequence a new XIFI Neutron plug-in should be released to accompany that OpenStack version.

4.8 Installation Manual

4.8.1 Installation Requirements

The following table summarizes the system requirements for healthy running the XIFI Neutron plug-in.

Operating system	Ubuntu 12.04 or higher
Memory	4 GB RAM
Hard drive	80 GB available in the hard disk
Python	Version 2.7

Table 19: System requirements for running the XIFI plug-in

4.8.2 Software Repository

For the time being, the software will be delivered via a Dropbox folder. It is foreseen to host the software in the project SVN and in a GIT repository once the software is available as open source software.

The public Dropbox link where the component can be found is the following:

https://www.dropbox.com/s/d78kgo89mtnmj9o/TID_xifi_package_v1.0.zip

4.8.3 Setup Guidelines

Once having setup the ABNO component, at the root of the installation package execute the following instruction:

```
./init_setup_2.sh -Plugin
```

Openstack has to be installed previously on the same host.

4.9 User Manual

The file ./plugin_config_file.xml has to be configured with the IP and port of the ABNO.

5 SITUATION AT M12

This section reports the real status of the activities related to the XIFI Network Controller at M12.

5.1 Network control tests performed

During this period two comprehensive tests have been accomplished to check the functionality of the XIFI Network Controller. The following sections describe in more detail the set-up and the results of these tests.

5.1.1 L2VPN connectivity test through the international network of Telefónica

In order to emulate a real, operational multi-site connectivity environment a test-bed was implemented in the labs of Telefónica Global Solutions, the international carrier company of the Telefónica group. The setup is illustrated in the following figure.

A datacentre is built with an OpenFlow-enabled switch and a server. The switches constitute the demarcation points. The connections between the switches are overlay paths. In the proof of concept a layer-2 VPN connection among data centres has been selected. The switches used are HP 5406zl, supporting OpenFlow 1.0.

Three servers (2 HP Proliant DL 380 GS and Dell Poweredge R210) are used to host both the software components described later and the VMs instantiated by the end users. The servers are directly attached to physical switches via point-to-point links.

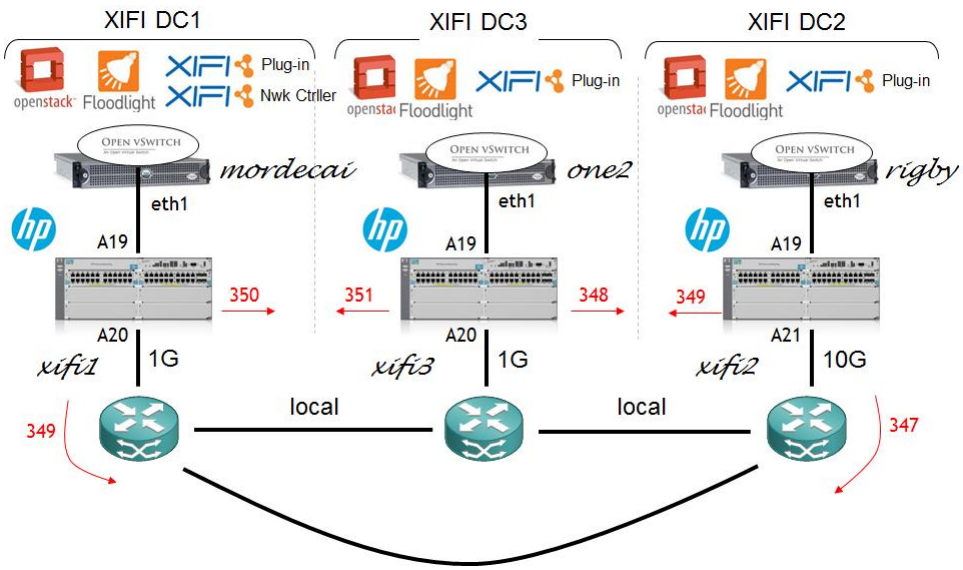


Figure 13: XIFI Network Controller test setup for L2VPN connectivity at Telefónica premises

The setup involves four main software components. The majority of these components are locally deployed in each of the data centres to manage and control the computing resources. Only the XIFI Network Controller, acting as network orchestrator, is centralized, as it has the whole view of the federation. Up to three data centres have been emulated by connecting one server per each physical OF switch, via GbEth ports. The connection set between data centres is composed by two links local to the lab in Madrid, plus an international Madrid – Paris – Frankfurt – Madrid link.

A traditional MPLS layer-2 VPN service provides the inter-DC connection. Dedicated VLANs are

defined for each data centre to distinguish the different connections. These VLANs are configured in the routers where the OF switches are connected.

The servers also run OpenStack as the cloud computing management software. An instance of Open vSwitch is created in the Openstack's Compute to build the internal connections between VMs. The virtual switch will be part of the topology reported by the local SDN controller, together with the physical switch acting as the demarcation point.

Cloud computing manager

OpenStack Grizzly is the software chosen for handling the computing resources. There is an independent OpenStack instance per data centre. All the services (Nova, Neutron, Cinder, Glance and Keystone) run in the same physical machine. According to OpenStack terminology, a server which is able to run virtual machines is called a "compute node". Thus, in the experiment, there is a single compute node per data centre.

OpenStack's Neutron component is in charge of providing the local overlay networks. It works with a technology dependent plugin. For XIFI project a customized plug-in has been developed to manage the internal DC connectivity. The plug-in interacts with the XIFI Network Controller which configures the Open vSwitch of each compute node. The XIFI Neutron plug-in will be released as Open Source in the framework of the XIFI project.

Local SDN Controller

An independent SDN controller is installed in each of the servers to emulate the SDN domain separation. The SDN controller used in each of the DCs for this test has been the Floodlight open source controller (version 0.90). Apart from injecting OpenFlow forwarding rules to the (virtual and physical) switches, the local SDN controller discovers the topology through the exchange of LLDP messages between the switches.

Network orchestrator

The XIFI Network Controller software modules are a specific development for the XIFI project, implemented in Java. The XIFI Network Controller will be released as Open Source software as a XIFI outcome. For this test a lite_NaaS component was developed as an interim step to a full integration with OpenNaaS. This lite-NaaS component is named ASO (Application Service Orchestrator) as described in chapter 3. This component will be substituted by OpenNaaS in the near future.

The routing and orchestration in the federated environment is achieved by the joint action of the ASO and the ABNO components. Two different kind of forwarding rules have to be populated to the switching elements for building the links end to end across the federation. One of them univocally connects each pair of VMs being part of the same logic network. The other one is due to the necessity of handling the broadcast traffic generated by each of the VM (e.g., ARP traffic).

5.1.2 Local point-to-point connectivity test in Trento node

The XIFI Network Controller has been tested in Trento during an activity related to a showcase called "*Quality of Experience in NaaS*" in WP6, reported in the deliverable D6.2. The architecture is composed of two servers with a single Ethernet connection between them, as depicted in the following picture.

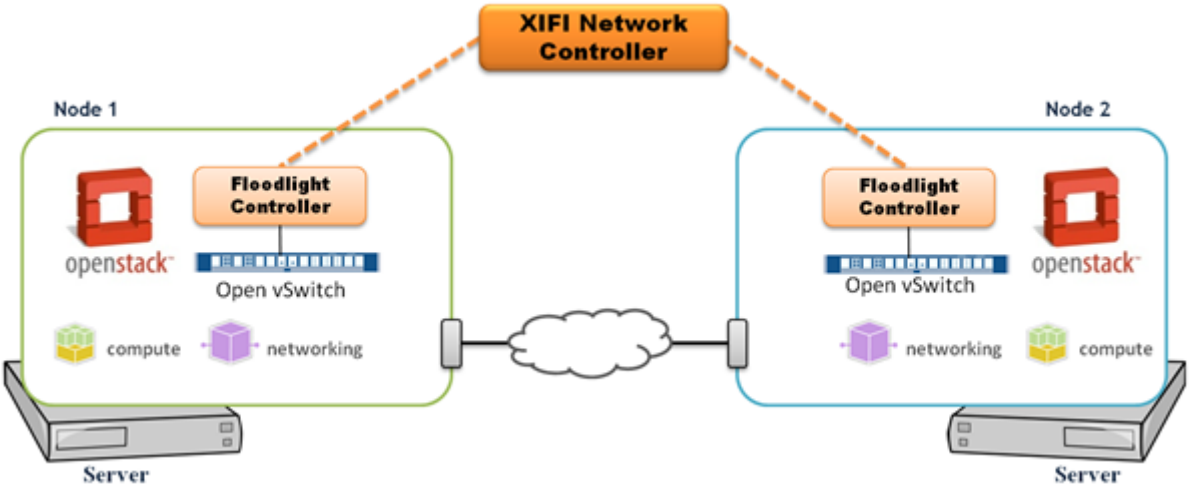


Figure 14: XIFI Network Controller test at Trento node

The environment is composed of two Dell PowerEdge T110 II Servers, with the following configuration:

- Processors: 4x3.10 GHz, CPU Intel Xeon E3-1200
- Memory: 8.0 GB DD3 at 1.60GHz
- Disk: 1TB SATA 7.200 rpm
- Interfaces: 3 Gigabit Ethernet NICs
- Operating system: Ubuntu 12.04 LTS

To deploy the correct OpenStack configuration, using the required Floodlight Neutron plugin, DevStack has been used. In particular we adopted the "All-In-One" configuration: this means the all OpenStack components are set up in a single machine. We obviously focused on Compute and Neutron: the first is fundamental to run Virtual Machines, the latter to grant connectivity between them.

During the first stage of tests we made sure that the XIFI Network Controller met the requirements. Once the configuration was up and running, we accomplished the target following the configuration proposed in section 3.9.2. As a result it was possible to enable connectivity between two virtual machines running in two separated OpenStack instances: the first one hosted in node 1, the second in node 2.

In particular it was possible to set up a video streaming application as reported in the mentioned showcase. To sum up, a VM acting as a Media Server (MS) streams the video to another VM acting as a Proxy Server (PS). The MS and the PS are located respectively on *node 1* and on *node 2*. Then the PS redirects the streaming video to a client (not depicted in the figure). The point is to simulate the connection between the XIFI nodes managed by means of the XIFI Network Controller and to use the functions of the controller to offer QoS, hence a QoE for the end user. Due to the current limitation of the PCE, that does not support redundant links, we performed the tests using ad-hoc script in order to simulate the fault of the primary path and switch to the backup path.

The main issue identified using Floodlight is the absence of L3 functionality. In fact, Floodlight uses MAC-based layer 2 network isolation and only the proprietary BigSwitch controller implements the L3 functionality. This also implies a scalability issue: each virtual machine requires unicast rules for

each virtual machine in the same network, which requires a high number of OpenFlow rules to be handled (both in the XIFI Network Controller and the OF switch).

The other issue is the lack of dynamism. For instances it is not possible to add a new virtual machine to the virtual network after the command *join_networks*. Moreover the configuration of the intra-domain and inter-domain connections requires manual configurations. This is maybe necessary for the inter-domain connectivity, but should be simplified considering the intra-domain case.

The tests did not consider the integration of the XIFI Network Controller with the OpenFlow enabled switches. In fact in Trento the two OF enabled switches adopted for XIFI are already installed in the production environment.

From the point of view of the pure network connection the target is reached but considering the whole XIFI architecture the drawbacks outlined above will have to be addressed.

5.2 Lite-NaaS component

As previously mentioned, an additional module that complements ABNO has been developed to provide service awareness in the first stage of the XIFI Network Controller development. The Application Service Orchestrator (ASO) is used to maintain the notion of connectivity service end to end by associating VMs located in separated data centres with the required network connection among them.

Once the integration with OpenNaaS is finished, this module will be substituted by OpenNaaS.

5.3 Floodlight SDN controller

Each data centre is considered to be an independent SDN domain. Every domain has its own controller, with control responsibilities over physical and virtual network resources within the data centre.

The deployed controller is Floodlight. Only the functionalities of pushing the forwarding rules and discovering the topology information are used from the controller.

Some limitations in the capabilities of Floodlight (mainly, L3 functionality) makes necessary to substitute this controller with another more powerful alternative. Chapter 6 elaborates more on this.

© 2013 - 2015 XIFI Consortium Parties

6 SHORT TERM EVOLUTION

This chapter introduces the current activities being carried out in Task 3.1 which will impose some evolution steps for the XIFI Network Controller in the short term. Two kind of actions are being taken, one of them referred to the internal evolution of the Network Controller, and the other one related to the external interaction with other components that are part of the XIFI project.

6.1 Development of capabilities internal to the controller

The XIFI Network Controller is a modular software element. This fact permits that every specific component can take its own independent evolutionary path, minimizing the impacts on the rest of the components comprising the XIFI Network Controller. Additionally, new components can be integrated in the overall XIFI Network Controller by defining integration interfaces with the rest of the modules.

This section describes the modules and capabilities currently being developed and integrated in the XIFI Network Controller.

6.1.1 OpenNaaS integration

OpenNaaS is one of the original software components considered in the XIFI Network Controller design. Its capabilities were already reported in Deliverable 3.1.

At this moment, the integration between OpenNaaS and the ABNO part of the XIFI Network Controller is being defined. Additionally, some functionalities to be covered by the OpenNaaS component are being discussed.

Essentially the OpenNaaS component will maintain the service awareness in the XIFI Network Controller architecture. It will keep information related to the resources associated to tenants on a per DC basis, including some identifiers that allow connectivity service tracking. For instance, data like MAC addresses of the VMs, ports where the VMs are attached to the Open vSwitch virtual device, local identifiers of the network in each DC assigned by the local instance of OpenStack, etc. Some other data complements the local information obtained from the resources deployed in each DC, and it is essential for the connectivity service, like the regions where the tenants have deployed their resources, or proper identifiers of the end-to-end connectivity service.

All of this information requires the development of specific data bases with appropriate indexing mechanisms for easy tracking of the service. This is essential not only for connectivity service creation, but also for modification of even deletion of the service. The information should be updated dynamically, according to the real needs of the tenant.

Finally, the workflows for the creation, modification and deletion of the services are also being developed. These workflows have to consider both the provision requests from the portal (or north bound interface for OpenNaaS) and the workflows already defined in the ABNO component (or south bound interface of OpenNaaS). Future triggering of connectivity service from a specific instance of OpenStack in one DC will need to be borne in mind when defining the north bound interface for an easy integration of this functionality in the future, if finally introduced.

6.1.2 QoS capabilities

One of the key differentiators of XIFI respect to other similar cloud community projects or commercial proposals is the capability of controlling the underlying transport network to a greater or lesser extent. This is partially done by the potential traffic engineering capabilities performed by the

PCE module of the XIFI Network Controller.

However, to fully provide programmatic QoS capabilities to the tenants, the XIFI Network Controller currently lacks of appropriate API mechanisms for that.

A first driver for enabling QoS is the work in the "Ofertie" use case. Nevertheless, enabling QoS in the XIFI Network Controller is considered as a key feature of the Network Controller.

The "Ofertie" use case intends to deploy on XIFI infrastructure on-line game servers being accessed by tenants as users of the service. Due to the requirements of QoS for this kind of service, the XIFI Network Controller should support QoS and network optimization capabilities.

By now the strategy under consideration is to directly expose on the XIFI Network Controller API generic QoS handling capabilities that can be directly mapped to the local SDN controllers in each DC. In principle these capabilities would allow the configuration of the queues of the ports in the OF-enabled switches (either physical or virtual) accomplishing traffic shaping and providing committed rate guarantees in the access.

Some experiences have previously been reported for the Floodlight SDN controller [6] and even for the Ryu controller [7] based on the previous ones.

Further integration with QoS capabilities in the transport between DCs is for further study, and will largely depend on the QoS capabilities that may be made available in the MD-VPN solution by Géant. Additionally, it is left for further study the combination of the flow monitoring capabilities already integrated in the XIFI Network Controller with the QoS capabilities mentioned here.

6.1.3 New local SDN controller

According to the control architecture in XIFI, the XIFI Network Controller instructs the local SDN controller in the different regions or DCs to populate the forwarding rules in the OpenFlow-based switching infrastructure to build the connectivity among VMs.

The SDN controller of reference for the XIFI architecture is Floodlight [6]. This is an open source controller in wide use, and was considered to be the local controller because of its wide use and extensive documentation.

However Floodlight has some drawbacks. Specially it lacks of capabilities for handling L3 as reported here http://docs.openstack.org/grizzly/openstack-network/admin/content/install_quantum-l3.html). The commercial counterpart, the Bigswitch controller incorporates all of the necessary L3 functionalities, and it is not foreseen that they will transfer to the open source version. Since the XIFI solution has to include full L3 capabilities, a reconsideration of the base local SDN controller has to be made.

Considering that all XIFI nodes have to use Grizzly, in the short term upgrades to Havana are not foreseen, and the deployment is almost done by the initial 5 nodes, it is important to select a new controller that handles these requirements:

- Integration and Support for all the Neutron functionalities
    - L2, L3, namespaces, tunneling (e.g. GRE, VxLAN)
- Seamless Integration with the current deployment
    - Grizzly release
    - The adoption of the new plugin should not be disruptive for the Infrastructure Owners

Moreover it is important to consider that at the moment almost all nodes are adopting the standard Open vSwitch plugin using GRE tunneling. This configuration is easy to install and does not require an additional NIC for the OpenStack Data Network (required in case of VLAN segmentation).

So the alternative to Floodlight has to satisfy the above-mentioned requirements.

At this point there are two possible alternatives under study, namely Ryu and OpenDayLight (ODL).

Ryu is an open source SDN controller based on Python which presents full L2 and L3 capabilities. It can play the same role as the Floodlight controller, and it offers similar mechanisms for integration, with a well-documented API.

On the other hand, ODL is an initiative promoted by the Linux Foundation. ODL is much more than a SDN controller since it incorporates many more functionalities which permit a more augmented control of the underlying network. ODL could be positioned in the future as the generic orchestrator of a connectivity service playing a similar role to the XIFI Network Controller today. Some of the components of the ODL architecture can be mapped (maybe not totally) to the ABNO building blocks in the XIFI Network Controller. By now ODL is not mature enough to do so, but one of the activities in progress in T3.1 is the monitoring of ODL as a community development alternative for the future. The other drawback of ODL is that it does not support the integration with Grizzly, the OpenStack release adopted in XIFI. This is not a point to underestimate because in the OpenStack release supported by ODL (called "Havana") some components that have been adopted by XIFI (e.g., the Open vSwitch plugin) are deprecated. In fact, in Havana there is a new plugin called Modular Layer 2 (ML2) that replaces the old plugins (https://wiki.openstack.org/wiki/Neutron#Havana_Release_and_ML2_Plugin_Update).

That being said, the most promising alternative is to migrate from Floodlight to Ryu, and this effort is part of the current work in T3.1. This can have further implications such as the need for adapting the flow monitoring capabilities from Floodlight to Ryu. This is left for further study.

6.2 Integration with elements external to the controller

This section addresses the activities being currently done in the integration process with external capabilities external to the XIFI Network Controller.

6.2.1 Connection to MD-VPN transport infrastructure

The XIFI deliverable 5.2 reports the XIFI backbone implementation for the initial nodes of the federation. The backbone connectivity is based on the MD-VPN transport service offered by the GÉANT community. The purpose of the XIFI Network Controller is to provide an orchestrated connectivity between the edges (i.e., the XIFI nodes or DCs) by stitching that local connectivity to the XIFI backbone. The XIFI Network Controller will instruct the local SDN controllers to compose the service by pushing the appropriate OF rules in the devices, either virtual (e.g., those created in the servers by OpenStack) or physical (e.g., those acting as demarcation points before entering the NREN infrastructure).

Then it is essential to integrate the local connectivity with the backbone transport. Due to the diversity of topological options it is required to clearly identify the service requirements to find commonalties and identify particularities for each of them.

Up to now, three potential cases have been identified, as shown in the figure below.

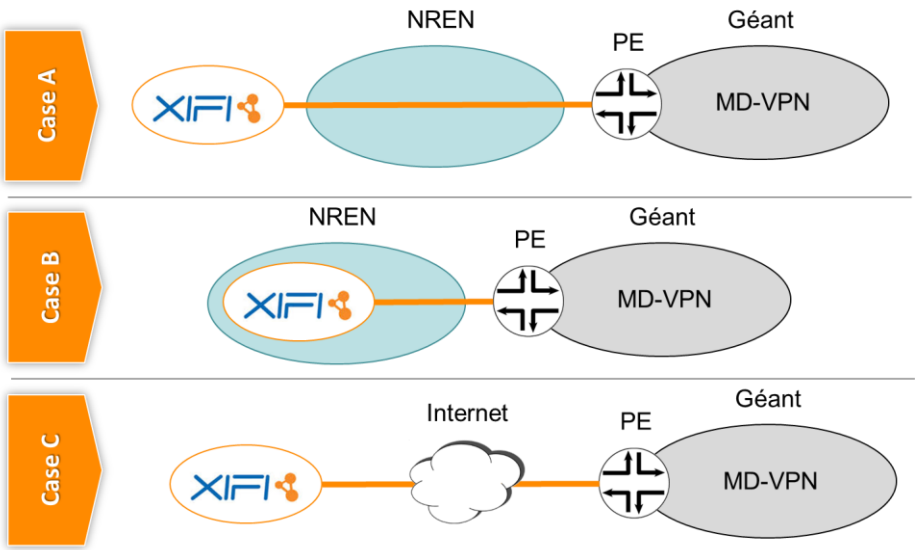


Figure 15: Integration scenarios with MD-VPN

The figure represents three potential cases:

- The XIFI node is external to the NREN which provides connectivity to the GÉANT MD-VPN service.
- The XIFI node is part of the NREN's infrastructure.
- The XIFI node gets access to the MD-VPN transport service through the Internet (by using some tunneling mechanism).

Currently the real cases for XIFI nodes connectivity to the MD-VPN are being defined. Forthcoming work will be to identify the service requirements (as mentioned before), to provide the functional definition of the integration with the MD-VPN service, and to propose a Proof of Concept for that.

6.2.2 Integration with the FI-Lab portal

The integration with the FI-Lab portal will be explored as consequence of the 3-Tier use case reported before. In this use case the workflows considered in the portal and in the XIFI Network Controller for connectivity between separate DCs will be aligned and tested.

No major impact is expected.

REFERENCES

[1] D. King, A. Farrel, "A PCE-based Architecture for Application-based Network Operations", draft-farrkingel-pce-abno-architecture-07 (work in progress), February 2014.

[2] OpenNaaS: http://opennaas.org/

[3] I. Bueno, J.I. Aznar, E. Escalona, J. Ferrer, J.A. García-Espín, " An OpenNaaS based SDN Framework for Dynamic QoS control," SDN for Future Networks and Services (SDN4FNS), (November 2013) http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6702533

[4] J.I. Aznar, .; Jara, M. ; Rosello, A. ; Wilson, D. ; Figuerola, S." OpenNaaS Based Management Solution for Inter-data Centres Connectivity, " IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom), December, 2013. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6735399

[5] XIFI Deliverable D3.1 "XIFI infrastructure adaptation components API open specification", http://wiki.fi-XIFI.eu/Public:D3.1

[6] http://www.openflowhub.org/display/floodlightcontroller/How+to+implement+Quality+Of+Service+using+Floodlight

[7] https://github.com/hu6360567/ryu-qos

APPENDIX A: INTERACTION WITH FLOODLIGHT AS LOCAL SDN CONTROLLER

The XIFI Network Controller orchestrates local SDN controllers, one per each XIFI node. The reason for that is because each XIFI node forms a separated SDN domain.

The Floodlight Open-Flow controller has been considered as the local SDN controller for the first release of the XIFI Network Controller. This Annex collects a number of queries/interactions used for the communication between the XIFI Network Controller (the Provisioning Manager module in the ABNO component) and the Floodlight controller.

Floodlight API is documented in http://www.openflowhub.org/display/floodlightcontroller/Floodlight+REST+API.

Some useful queries are the following:

- List all the switches whose controller is the floodlight at $CONTROLLER_IP and port 8080

```
curl http://$CONTROLLER_IP:8080/wm/core/controller/switches/json
```

- Retrieve topology:

```
curl http://$CONTROLLER_IP:8080/wm/topology/links
```

- Clear all rules from all switches:

```
curl http://$CONTROLLER_IP:8080/wm/staticflowentrypusher/all/json
```

- Push a flow

```
curl -d '{"switch": "00:00:00:00:00:00:00:01", "name":"flow-mod-1",
"priority":"32768", "ingress-port":"1","active":"true",
"actions":"output=2"}'
http://$CONTROLLER_IP:8080/wm/staticflowentrypusher/json
```