OFELIA

ICT-258365

Deliverable 5.1

1st Version of the OFELIA Management Software

Editor:	*Marc Suñé (Fundació Privada i2cat, Internet i Innovació Digital a Catalunya(i2cat))*
Work Package (leader)	WP5 (Marc Suñé, *Fundació Privada i2CAT*, Internet i Innovació Digital a Catalunya (i2CAT))
Deliverable nature:	Prototype (P)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	28/02/11
Actual delivery date:	07/03/11
Suggested readers:	t.b.d.
Version:	1.0
Total number of pages:	63
Keywords:	OFELIA-FP7 WP5 D5.1

Disclaimer

Imprint

[Project title]	***OpenFlow in Europe – Linking Infrastructure and Applications***
[short title]	***OFELIA***
[Number and title of work package]	***WP5 – Software Development***
[Document title]	***D5.1***
[Editor]	***Marc Suñé, Fundació i2CAT***
[Work package leader]	***Marc Suñé, Fundació i2CAT***
[Task leader]	***Marc Suñé, Fundació i2CAT***
[PM (estimated)]	***30***
[PM (consumed)]	***Number***

Copyright notice

Executive summary

The main goal of the OFELIA WP5 is to provide the required software to support the operation of the facility. Based on the different requirements for design, management and implementation , several different technologies, tools and existing testbed control frameworks have been studied. This document provides a description of the work that has been carried out during the first period of the task T.5.2, which concludes with the release of the first version of the OFELIA Control framework.

This document provides a brief overview of the state-of-the-art study that was done in task T5.1, (see document MS 5.1 [21] for details), which evaluated and tested several existing control frameworks and technologies that could potentially simplify and improve the process of OFELIA 's control framework development. Based on the conclusions drawn in [21] , OFELIA will adopt the Stanford's control framework tool (called Expedient and Optin Manager) as the base implementation code, and which its functionality has been shown in various US OpenFlow GENI testbed demonstrations beds various GENI demonstrations [22][23].

Furthermore, this document provides an overview of the principles and basic objectives that were pursued during this first phase of the development, as well as a description of what has been defined as the basic use case for phase 1. In this “basic use case” document, the basic interaction of the user with the OFELIA facility is synthesized. This enables extracting the basic requirements of the software, which will evolve as the facility receives new users and projects.

The remainder of this document focuses on a brief overview of the Stanford's control framework (i.e. Expedient and Opt-in Manager tool) including a brief introduction to its architectural design and coding structure.

Section 3 offers a detailed description of the work that has been carried out in WP5 during T5.2 including a description of the scope of each of the subtasks, the challenges that are being faced, tools and technologies used, as well as architecture and coding strategy of the proposed solution. In addition Section 4 exposes the current status of each of these subtasks as well as a brief description of the next steps for the following months

Finally, as part of the deliverable, a set of guidelines and comments on the software bundle that can be found and downloaded for evaluation in the OFELIA FP7 repositories is given in section 5.

List of authors

Organisation/Company	Author
NEC	Thomas Dietz
UESSEX	Jayakumar Ramanujam
UESSEX	Siamak Azodolmolky
UESSEX	Reza Nejabati
UESSEX	Dimitra Simeonidou
ADVA	Pawel Kaczmarek
ADVA	Pawel Kostecki
ETHZ	Jose Francisco Mingorance- Puga
ETHZ	Wolfgang Mühlbauer
EICT	Andreas Köpsel
EICT	Tom Rothe
IBBT	Didier Colle
i2CAT	Leonardo Bergesio
i2CAT	Marc Suñé
i2CAT	Alejandro Chuang

Table of Contents

List of figures and/or list of tables

Abbreviations

AA	Authentication and Authorization
AID	Access Identifier
AM	Aggregate Manager
API	Application Programming Interface
CLI	Command Line Interface
CM	Component Manager
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DN	Distinguished Name
DWDM	Dense Wavelength Division Multiplexing
E-GENI	Enterprise GENI
FIRE	Future Internet Research and Experimentation
GCF	GENI Control Framework
GMPLS	Generalized Multi-Protocol Label Switching
GPL	General Public License
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
IM	Island Manager
IP	Internet Protocol
LDAP	Lightweight Directory Access Protocol
LVM	Logical Volume Management
MAC	Media Access Control
NOC	Network Operations Center
OAM	Oracle Applications Manager
OF	OpenFlow
OF	OpenFlow
OFELIA	OpenFlow in Europe – Linking Infrastructure and Applications
OS	Operating System
OSC	Optical Supervisory Channel
OSPF-TE	Open Shortest Path First-Traffic Enginyeering
ROADM	Reconfigurable Optical Add-Drop Multiplexer
ROADM	Reconfigurable Optical Add Drop Multiplexer
RPC	Remote Procedure Call
RSpec	Resource Specification
SDH	Synchronous Digital Hierarchy
SDK	Software Development Kit
SFA	Slice-based Federation Architecture
SNAC	Simple Network Access Controller
SONET	Synchronous Optical Network
SSH	Secure Shell
STS	Synchronous Transport Signal
TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
UI	User Interface
UID	User Identifier
URL	Universal Resource Locator
UUID	Universally Unique Identifier
VCG	Virtual Concatenation Groups

VCG	Virtual Concatenation Group
VLAN	Virtual Local Area Network
VM	Virtual Machine
VPN	Virtual Private Network
VT AM	Virtualization Technology Aggregate Manager
WP	Work Package
XML	Extensible Markup Language

1 Introduction

The objective of this deliverable "D5.1 - 1st version of the OFELIA management software" and in particular of the Task 5.2 within WP5 work package, is to provide an implementation of the facility control framework, to try to automate as much as possible the operation and maintenance of Openflow [1] islands.

1.1 OFELIA Control framework overview

1.1.1 The objective and development principles

The OFELIA Control Framework can be defined as the control plane application for OFELIA -FP7 facility. The main purpose of the framework is to automate, simplify and authorize users to create network slices and deploy resources available within OFELIA islands for various types of experimental projects.

Task 5.2 is dedicated towards design and implementation of the OFELIA control framework and is split logically into three distinct phases:

1. Phase I will have the commitment to code the first early version of the control framework, which will be focused on the management of the island's local resources. No inter-island resource control will be available in Phase I. However, and having in mind that Phase II will require the integration of the different islands, special attention will be paid to make the right design decisions to prevent conflicts, errors and re-engineering efforts in Phase II and Phase III.

2. Phase II, will be devoted to empower the control framework with mechanisms to allocate resources across multiple islands within the same project and slice. In addition, this phase will also include software improvements of some of the basic features implemented in Phase I, taking into account experiences acquired from the different internal and external facility users.

3. Finally, Phase III will continue the overall improvement of the control framework, especially taking into account the requirements, suggestions and comments inferred from users of the first open call.

From the above description one can see that the development of the control framework will be highly dependent of the user requirements, and hence it will continuosly adapted according to new requests and suggestions proposed by the facility users. In this sense, although at least one release of the software will be published per phase, T5.2 will try to follow a dynamic and on demand Agile software development approach, constantly evolving and improving the software.

From the point of view of the first phase, which will be the scope of this deliverable, Task 5.2 has taken into account the document MS2.1 "Report on initial requirement study and analysis of use-cases ready" (See [20]) that has been delivered from WP2, and fundamentally the "Basic Use Case" document as a result of the collaboration of WP2 and WP5, as well as some generic requirements that were deduced from the experiences on Openflow testbeds in USA (basically in the GENI project). An analysis of the so far collected requirements resulted in a set of design principles that define the basic requirements for the control framework, facility and the underlying physical network substrate.

The following principles have guided the development work for the OFELIA control framework:

- **Resource allocation:** the user should be able to allocate or book resources in an easy way. Within the OFELIA testbed the different basic types of resources will be; OpenFlow resources (such as openflow enabled switches, switch ports, traffic flows) , hosts: (either virtual or physical), in-cluster VMs (in the case of the IBBT virtual-wall), IBBT's WLAN testbed or any other resource that partners want to include in the OFELIA facility.

- **Experiment and project based resource allocation:** the resource allocation must be made per project and slice. A slice is defined as the smallest indivisible entity that is composed by the resources necessary to carry out an experiment. A project may be composed by one or more slices.

In this sense, and tightly related to the following requirement, the control framework has also the objective to isolate as much as possible each and every single slice from each other sharing the same infrastructure substrate. In the particular case of OFELIA, special attention needs to be paid to network traffic segregation between slices.

- **AA:** the control framework has to support user authentication and authorization mechanism. Users should have different levels of permissions based on their status, having at least one superuser or "root user" per island.

- **Usability:** users, in this case, experimenters should have access to a comprehensive and easy to use interface. In this sense, the preferred way of interacting with the users is a web-based interface. Special attention will be paid to try to bring to the user, as much as possible, a unified interface for managing everything related to the OFELIA facility.

- **Scalability:** the control framework must be scalable, in terms of number of users, number of supporting concurrent experiments and number of managed resources.

- **Island autonomy:** one of the basic requirements that the control framework will have to deal with that some partners have expressed, is that the control framework should be able to manage resources locally (in the island) even if connection with the rest of the islands is lost. Therefore effectively being completely autonomous.

- **Robustness and stability.** Stability and robustness is a must.

- **Monitoring:** the control framework should perform monitoring tasks, for both the components conforming the control framework, and the resources of the testbed.

- **Efficiency.** The development will try to be as efficient as possible in terms of coding and trying to reuse as much as possible the different open-source tools and libraries that the community offers , to focus their efforts on the development of those aspects that are particular for OFELIA control framework and also to improve those open-source libraries and tools.

1.1.2 MS5.1 Initial study of the state-of-the-art. Starting point.

During task T5.1 an in-depth study of the state of the art of several control frameworks, technologies and tools for managing testbed resources was conducted. The milestone document MS5.1 "Software development environment setup" [21] contains all this information, and also an analysis of each of these tools to be used in the specific case of OFELIA facility. In order to

improve implementation efficiency, and taking into account the tight time schedule for delivery of the Phase I software package, it was decided to adopt an existing control framework rather than implementing a new one. For improved stability and robustness existing mature (open source) software components should be used whenever possible resulting in a configuration above implementation approach or DRY concept (don't repeat yourself).

The conclusions drawn by MS5.1 can be basically summarized in the following statements:

- The E-GENI Expedient [2] tool, in conjunction with the rest of software (plug-ins, aggregate managers, etc) will be used as basis for the OFELIA control framework implementation.
- During Phase I of the project each island will work isolated from each other. Every island will have one instance of Expedient plus at least two aggregate managers: Opt-in Manager [3] for the OpenFlow resources, and the virtualization technology aggregate (which will be XEN for the Phase I [4]).
- The architecture that the OFELIA control framework will follow as much as possible will be inspired by the Slice-based Federation Architecture, SFA [5].
- The OpenFlow development effort regarding Expedient, the Opt-in Manager and FlowVisor will be conducted in collaboration with Stanford University.
- The code repository will be GIT and will be structured in specific branches.
- The basic programming language will be Python. The Python based Django web development framework is going to be used for the user web interfaces developments since it is the one in use in the current software. However, it will be up to the consortium to use other programming languages and technologies for new developments and the modifications carried out over the current implementation.

It is also remarkable that OFELIAWP5 has been in direct contact with the developers at Stanford, in order to coordinate development efforts towards a unified and improved control framework.

1.2 Basic use-case

During this period of time, and besides the document MS2.1, both WP2 and 5 have joined efforts to define an internal document to summarize the basic use-case that the OFELIAfacility is going to support on the preliminary phase. In other words, the basic use case defines the basic functional tests to verify proper operation of the OFELIA control framework, its OAM mechanisms, and the interface towards the user.

1.2.1 Description of the basic scenario for the use case

The following diagram shows what has been considered as the basic scenario for phase1. In this diagram the basic components from which all OFELIA islands should be composed are:

- OpenFlow enabled network substrate. An OpenFlow capable network must interconnect the different data plane elements inside the island (hosts, FlowVisor, etc and other control framework elements).
- Hosts. A set of hosts are supposed to interact among others. These hosts will have basically two roles:
    - Act as end-points, and *hence* sending and receiving traffic inside the slice. This will basically be done by using VMs inside one or more virtualized island servers. The impact of using virtualized entities as data sinks and sources in performance tests is for further study.

- Run the controller software. As the core functionality of OpenFlow is the split of data and control plane, experimenters will deploy own controller entities hosting a variety of network applications. The deployment of controller entities and how these entities interact cannot be foreseen by OFELIA. However, at least one controller per slice (could be more, depending of the configuration of the slice) is the minimum requirement. Further extensions may be necessary to this initial assumption in the next phases.

- **FlowVisor**. Each island will have at least one FlowVisor entity to which users will connect their controllers (e.g. NOX controller). As FlowVisor provides the core slicing functionality for isolating slices and experiments, FlowVisor access and the ability control the FlowVisor must be restricted to OFELIA's control framework. In this sense, the control framework must ensure proper configuration of FlowVisor, and hence enforce the isolation between isolation, and prevent misbehavior by the user, by means of an authorization and authentication module.

- **Control network.** The control network will be used by the different entities of the control framework and facility service hosts to exchange data in a secure way. In principle, there should be no restriction on having the control network and data network in the same logical network. However and for security reasons an out-of-band control network is preferred. Please note, that specific requirements may be defined by the individual islands based on their local deployments, e.g. L2 vs. L3 based core, availability of (dark) fibers when multiple sub-islands are going to be connected, requirements defined by existing test beds (optical devices in Essex, wireless devices in Berlin, etc.). The OFELIA control framework is designed and configured to be useful for many heterogeneous environments and to reflect future constraints defined by new islands.

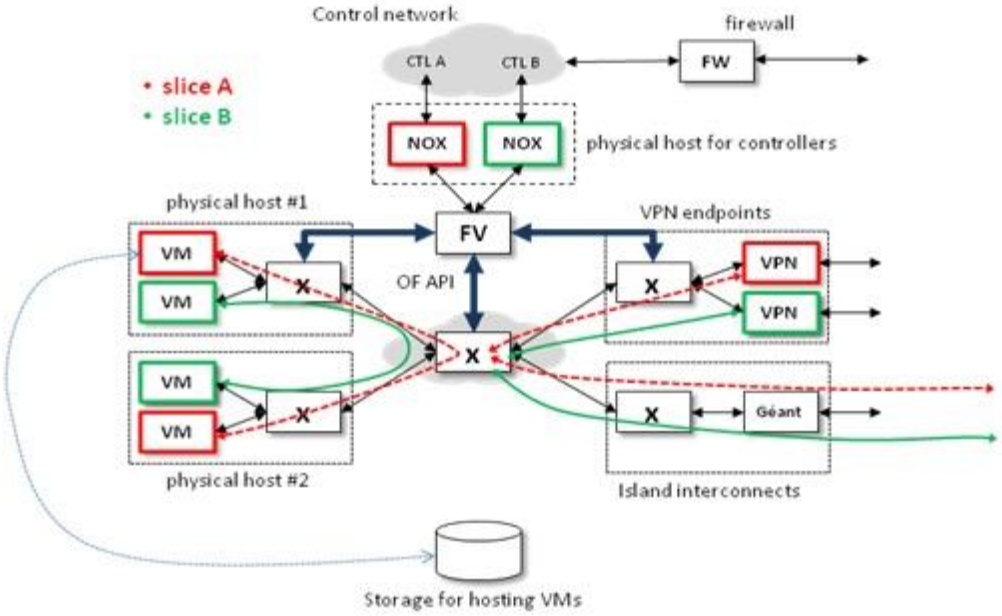


Figure 1: Isolated island basic scenario

1.2.2 Description of the basic use case.

The document of the basic use case can be found in Appendix A.

Please note that the slicing mechanism adopted by OFELIA is still under discussion at the time of writing this document. Several proposals have been made including MAC address based slicing, VLAN based slicing, IP based slicing.

1.3 General overview of the current results achieved.

The current status of the code can be browsed in the GIT repository of the project (See section 3.1.1.1) under the "ofelia-stable" branch. To summarize, the current implementation includes the basic functionality expected for phase I, namely: provisioning of XEN server resources (VMs) through the Expedient web interface, integration of LDAP and the automation of flow allocation (that is in process of being implemented). The software has been tested and is still under testing by some partners, specifically the GUI, and also other GUI interface aspects have been modified and customized.

It is important to remark that the adoption of Expedient and Opt-in manager as a basis for the implementation of the OFELIA Control Framework has obviously some advantages. These advantages are basically that a part of the features that were planned to be developed for the OFELIA tool, have been already coded, and also the fact that the collaboration between both OFELIA and Stanford University is beneficial for the open-source community and especially to other projects that might require testbed management solutions.

However, adopting an existing framework may also result in a number of less beneficial issues: . A significant learning curve is related to the existing code base and the technologies used. Furthermore, the fact of being a solution for another test bed (although its purpose might have some similarities with OFELIA), does not fit perfectly on the particular requirements defined within OFELIA. Therefore enhancements, modifications and extensions have to be implemented.

In addition, the Expedient used as a basis for the implementation work in OFELIA, has never been used in a production environment, and has only been tested in some demonstrations and low-scale user testing, hence can be categorized as a pre-production application. As the expedient code is still alpha quality software, significant efforts must be invested for testing the framework, fixing potential bugs and stabilizing its features. From that perspective expedient generates a similar work load compared to a self-development.
A detailed description of the current status of the development can be found in section 4.

2 Base software

2.1 Expedient tool

Expedient is a pluggable centralized GENI control framework. It is inspired by travel websites that allow a user to book a flight, hotels, and rental cars all within the same system. It is implemented as a Web application using Python and the Django Web framework. It provides simple abstract classes that resource developers can extend to build a plug-in for their resource types. It provides project, slice, and user management so that they do not have to worry about it.

Expedient tool is still a proof-of-concept that yet has not been used in real production context. It is released under an Open Source License and less restrictive than GPL.

Expedient tool tries to implement all the requirements for a GENI control framework. Its design is based on the following core functionalities:

- **Simplicity and Extensibility**. It has been designed with the purpose of being improved by the addition of new plug-ins capable of handling different kind of resources. In this sense it is positive that each plug-in is independent inside Expedient and different plug-ins for different resources can be developed separately.

- **User convinience.** Expedient enables rich user interfaces with its pluggable architecture. It allows developers to write user interface plug-ins that are tailored to sets of resource types at a time.

- **Security and Reliability.** Users authenticate themselves with Expedient and Expedient acts as a gateway for all their transactions with resources. If a resource provider does not want to implement authentication and authorization for each user who uses its resources then it can delegate these functions to Expedient.

Expedient's architecture is based on a central control block from where the different plug-ins are connected to the correspondent Aggregate Manager (AM) (see Figure 2). Each Aggregate Manager will be responsible for the management of the resources that are underneath it and that will be presented to the user through the Expedient user interface in a homogeneous way.

Core element in the expedient architecture is the aggregation manager. Typically, all physical resources are controlled by some kind of management framework and interface (e.g. virtual machine monitors control the operation of virtual servers and the surrounding physical environment). The AM uses the resource's native management interface for configuration and monitoring and exposes this interface via an abstract interface towards the expedient control framework.

As mentioned before, Expedient is a Django based web application (Django is a Python based web development framework) and follows a modular approach, i.e. its functionality can be extended by writing and deriving new "plug-ins" from a set of base classes. Each aggregate manager is connected to a resource specific plugin within expedient, so the AM binds actually the resource's management interface to expedient via the plugin.

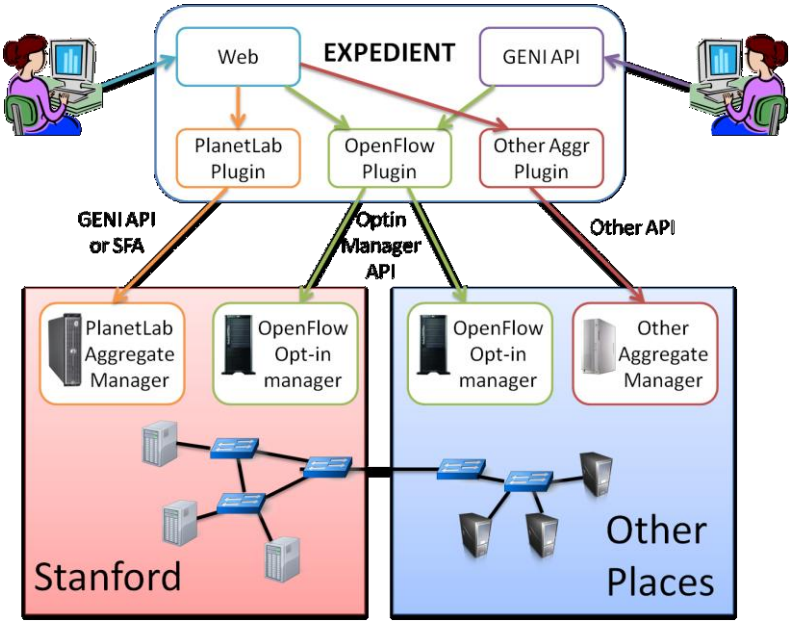


Figure 2: Expedient tool architecture.

There exists a one-to-one relationship between an aggregation manager and a corresponding plug-in within Expedient. Users with administrative rights may register new plug-in/AM pairs for controlling different resources via the Expedient web interface, thus Expedient is a modular framework with significant extensibility.

Expedient uses MySQL as persistent storage for user authentication and management. On one side, users authenticate themselves against Expedient and on the other side, Expedient authenticates itself against the AM database. Expedient runs on top of an Apache2 server.

The different blocks that take part in the communication between the user and the resources to be managed are communicated through XML-RPC protocol. XML-RPC protocol is a remote procedure calling that uses HTTP as the transport and XML as the encoding.

2.1.1 Code analysis

Expedient's code base consists of three main packages: clearinghouse, common and UI. The following figure shows a diagram of the package structure:

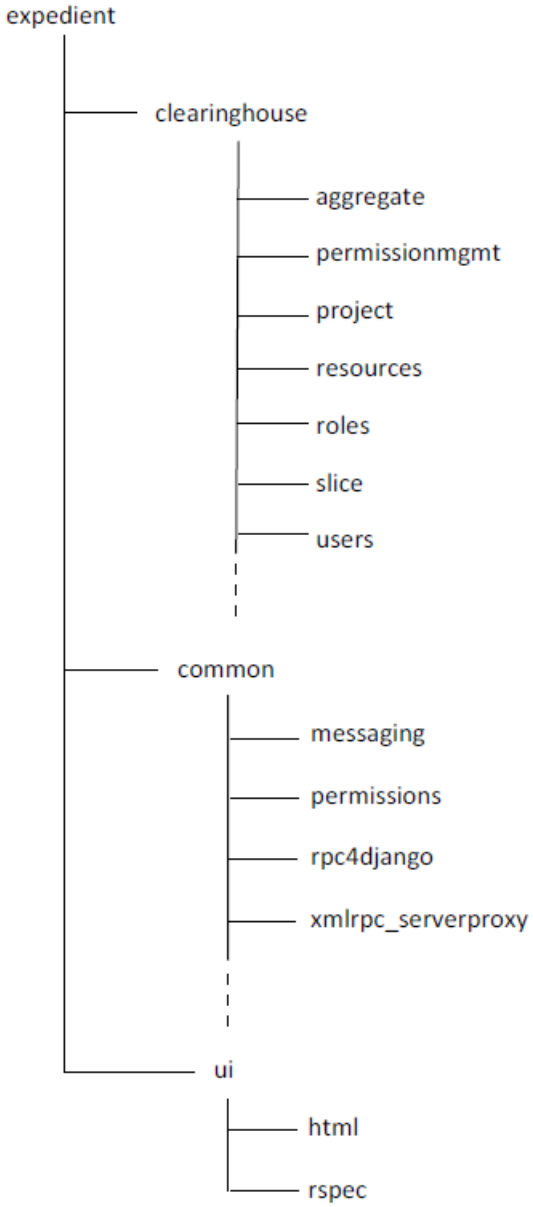


Figure 3: Expedient code structure.

In the `/expedient/clearinghouse` package we can find the modules that contain the main models that correspond to the information data to be stored in the database and managed by the tool. The main modules that can be found are the following:

- **aggregate:** manages all the actions that can be done over an aggregate[1]. For example, listing, deleting or adding aggregates to Expedient tool or to any project or slice. When adding new plug-ins to Expedient tool, the new aggregate model should inherit from the Aggregate model inside this package.
- **permissionmgmt:** handles all the permission requests and approvals or denials concerning users, projects and slices.

[1] Aggregate: Abstraction concept that represents a generic set of resources

- **project**: manages all the aspects related to the project such as the members that belong to it, shows and redirects to the slices and aggregates that are contained by it and allows to manage the roles of the projects members.
- **resources:** basically it contains the model that will represent the resources in Expedient.
- **roles:** handles all aspects related to roles such as creating, deleting or updating roles.
- **slice:** manages the slices of the project. It allows to manage the aggregates added to the slice, to start o stop the slice and redirects to the ui where the user will be able to manage the resources belonging to the slice.
- **users:** manages user issues such as listing all the users assigned to a project, showing a web formulary to add users to a project and saving, deleting, registering or activating users.

In the `/expedient/common` package we can find auxiliary modules. The main blocks are the following:

- **messaging:** manages all the messaging system prompted at the Expedient home page where the user can see all the actions that have been done regarding aggregates, projects, slices, etc.
- **permissions:** holds all the classes related to permission objects.
- **rpc4django:** contains the dispatchers that manage the communication responses.
- **xmlrpc_serverproxy:** implements a server proxy for XML-RPC calls.

Finally in `/expedient/ui` package we can find the modules that manage the user interface that the user sees when managing the resources in a particular slice. It contains the following blocks:

- **html:** manages the web interface that allows the user to manage the resources given a slice in a project.
- **rspec:** shows interface to download and upload RSpecs of the OpenFlow Manager.

Apart from the Expedient Tool's code we can also find several plug-ins that have been implemented. The OpenFlow package manages the OpenFlow plug-in that allows Expedient to manage OpenFlow network flows. The package `expedient_geni` holds the PlanetLab plug-in which has not been used in OFELIA facility [1].
In `/expedient/src/python/templates` package we can find all the templates that are loaded by the Expdient tool.

Regarding the code of the OpenFlow Aggregate Manager, the Opt-in Manager, we can find the following relevant packages:

- **admin_manager:** manages the user's flowspaces.
- **flowspace:** package that defines the flowspace class model.
- **opts:** handles the models that hold the information about verified flowspaces for each user and administrator and also the information about the topology and flowspace of an experiment.
- **users:** manages the information concerning the user's or administrator's profile.
- **xmlrpc_server:** implements a server proxy for XML-RPC calls.

2.2 Opt-in Manager

Opt-in Manager (OM) is the Aggregate Manager for OpenFlow resources. In a nutshell, Opt-in Manager is a database and web UI that holds information about the flowspace that each user owns and the list of experiments that are running in a network along with the flowspaces that they are

interested in controlling. The web UI allows users to opt their traffic into individual experiments. When a user opts into an experiment, the Opt-in manager finds the intersection of that user's flowspace and the experiment's flowspace and pushes it down to FlowVisor causing the packets matching the intersection to be controlled by the requested experiment's controller.

In the OFELIA facility, the web UI should be used by Island Managers rather than by users, since the selection of the flowspace and the experiments (projects, slices) is done through the OpenFlow plugin in the Expedient tool, which offers a web UI from for managing OpenFlow resources.

Opt-in Manager needs a way to know which user owns which piece of flowspace. Each user can request a piece of flowspace from an administrator who owns that piece of flowspace. At the install time, a root administrator will own the entire flowspace, and if any user requests a piece of flowspace, the request will be routed to the root administrator for confirmation. However, an administrator can delegate some portion of his or her flowspace to a lower-level administrator in which case that administrator will do the confirmation of flowspace requests for that portion. (one may think of this as the hierarchy of network administrators in a university campus, with a single administrator for the entire campus, several department administrators, etc.)

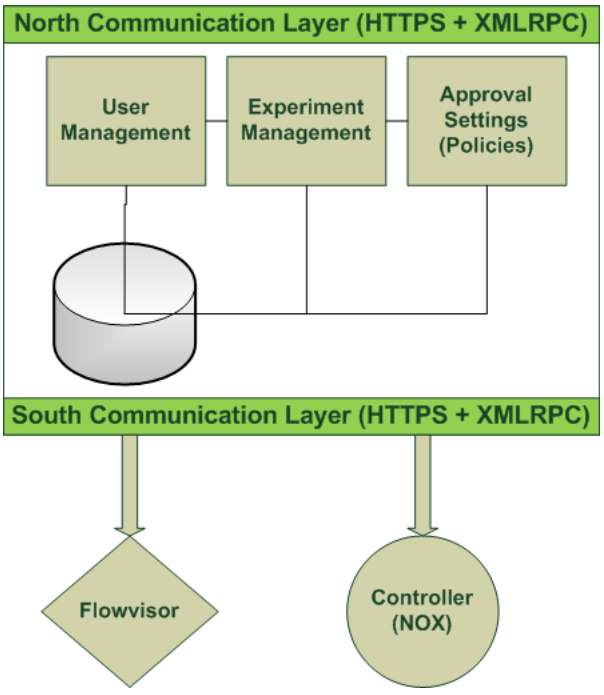
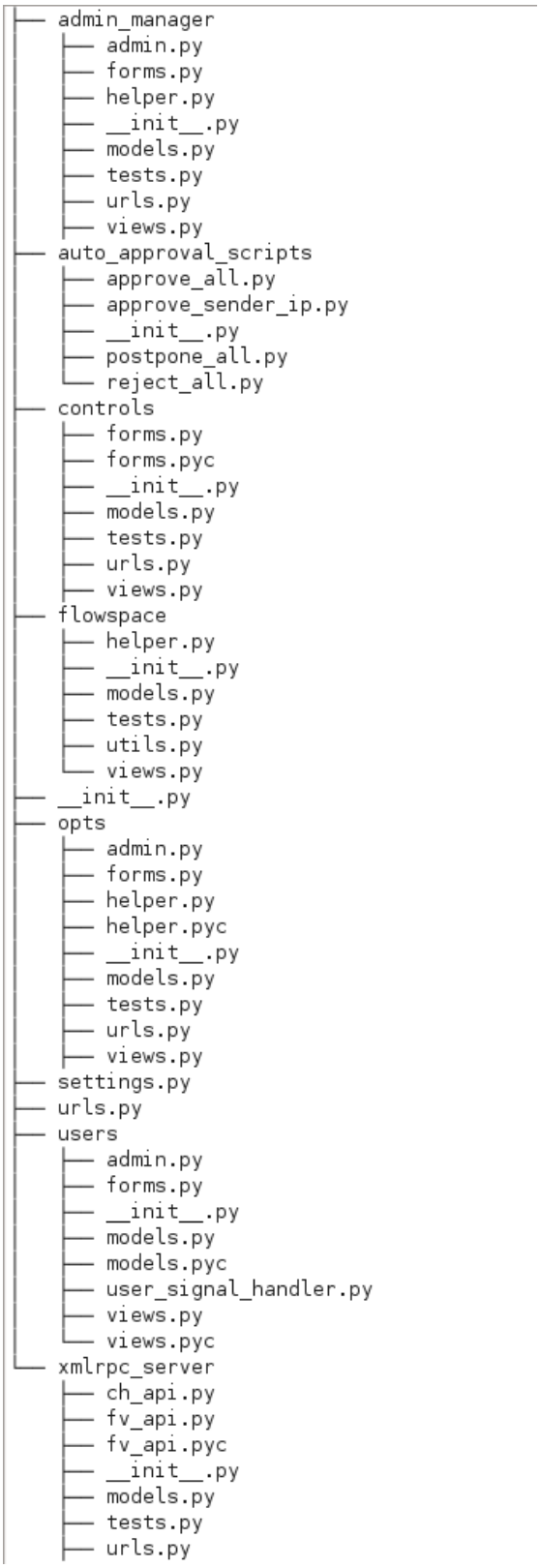


Figure 4: Basic Opt-in Manager architecture connected to FlowVisor and Controller

A basic architecture of the Opt-in Manager can be found in Figure 4 . The northbound communication layer is used to communicate the Expedient OpenFlow plug-in and the Opt-in Manager. The southbound interface is used for communicating the Opt-in with the FlowVisor and the controllers of the experiments.

2.2.1 Code analysis



- Admin_manager: administrator functions (promote user, manage flowspaces, etc.).
- auto_approval_scripts: python scripts defining auto-approvable flowspaces and actions to be done.
- controls: actions to set the clearinghouse user and the FlowVisor.
- flowspace: flowspace data model definition.
- opts: opts data model definition and management functions.
- users: users management
- xmlrpc_server: rpc methods definition that implements the interface exposed to Expedient and to the FlowVisor.

2.2.2 Adaptation and expansion of Opt-in Manager

For the OFELIA facility it should not be necessary to do many changes in the current Opt-in architecture and functionalities. Nevertheless, there are some points identified where improvements are required.

- Security and federation possibilities improvement: complete adaptation to SFA. Please note that adopting the SFA architecture for OFELIA is still under discussion.

- Improve the Policy Manager: There is an existent module in Opt-in where it is possible to set Auto-approval settings for the flowspaces defined by the users. However, it seems the current possibilities (Manual approve, Approve all, Approve sender IP, Reject All, Remote) are not customized enough. Therefore a better set of rules should be implemented. Moreover, despite the fact that the current implementation allows the administrator to create rule sets coding them, what would be necessary is a module in the web UI from where the Island Managers could be able to define rules and add them to the different profiles.

- Implement a callback function that notifies Expedient, that the physical OpenFlow topology underneath has changed.

3 Phase 1 development: OFELIA Control Framework

3.1 Overview of the development phase1

Phase 1 development has been split into several sub-tasks according to the different objectives that each of these sub-tasks had. Each and every single sub-task has worked in its own environment based on the stable version of the code, thanks, on one hand, to the GIT repository and, on the other, to the SDK appliance.

3.1.1 Development environment and SDK

This section provides a brief overview of the available infrastructure for management of developed software within the OFELIA project. The OFELIA control framework is designed to control and monitor the operation of the individual OpenFlow enabled islands. Within OFELIA existing software from other projects or programs (FIRE, GENI, etc.) as well as self-written software will be used. For storing forks of existing software and hosting own developments, a GIT repository [1] has been deployed.

- Repository address: ssh://firstname.lastname@alpha.fp7-ofelia.eu/home/ofelia/ofelia-git

Right now, this repository is for project internal use, i.e. access is restricted to project partners only. Additional repositories may be created for providing access for external users to the developed or extended software or within OFELIA.

3.1.1.1 Repository Access

The GIT repository is integrated with the OFELIA project infrastructure including authentication and authorization. Thus, for accessing the repository a user has to create an OFELIA project account first. The URL for obtaining a user account is :

- https://alpha.fp7-ofelia.eu/seradmi/register

A valid e-mail address is required as an activation code for this account to be sent to this address. Each registration will be manually reviewed by EICT to avoid repository access of unauthorized persons.

During registration, a user account is created in the OFELIA project LDAP directory with the following parameters:

Parameter	**Value**
cn	Firstname Lastname
objectClass	person, organizationalPerson, inetOrgPerson, posixAccount
homeDirectory	/home/firstname.lastname
sn	Lastname
uid	firstname.lastname

uidNumber	server internal Unix user ID (13xx)
gidNumber	server internal Unix group ID (1300)
Description	Description as defined by the user during registration
loginShell	/usr/bin/git-shell
mail	e-mail address as defined by the user during registration
o	Organization as defined by the user during registration
userPassword	Password as defined by the user during registration

The "mail" parameter is used as user identification on almost all services deployed on the OFELIA project server. **However, the username used for repository access is defined by the "uid" parameter and is constructed by "firstname.lastname".**

Note: A user may edit his account details any time via https://alpha.fp7-ofelia.eu/seradmi.

The user is automatically added to a project server internal git repository access group with the following DN:

- cn=ofelia-devel,ou=groups,ou=ofelia,dc=eict,dc=de

All access is via secure shell and with a restricted git-shell, i.e. users cannot log into the system as the git-shell only accepts git related commands.

For cloning the git repository for doing local development work, the following command is used:

- git clone ssh://<your_uid>@alpha.fp7-ofelia.eu/home/ofelia/ofelia-git

3.1.1.2 Development Tracker

A development tracker has been deployed for issuing bugs and requesting features. This way, it is possible to have a complete history of the code's changes and to track the development of the OFELIA control framework.

3.1.1.3 IDE, programming language and coding guidelines

The programming language decision is determined by the fact that Expedient has been used as a starting point for the development of the OFELIA control framework. As the former is built with Python over the Django framework and Apache server, the development will continue with this language in order to take advantage of the existing code and make the integration of the new code fragments easier.

In order to implement all the different sub-tasks, the code repository has been organized in different branches per task.

On one side, there is a branch for the stable version of Expedient. This is used, at the beginning, by the OFELIA team to get the original code to start developing. A second branch is used for the stable code of the OFELIA application and it contains the different releases. This way Stanford developers are able to get the stable OFELIA code from this branch, integrate it with their Expedient code and then the OFELIA team will be able to update its Expedient branch from the updated code.

Finally, there is a third branch for the in-development[2] code for OFELIA control framework. In addition, each specific task has its own branch of code to be merged in the in-development branch.

This organization allows an independent way for each task to move forward and at the same time it assures that not stable code mixes with the stable one.

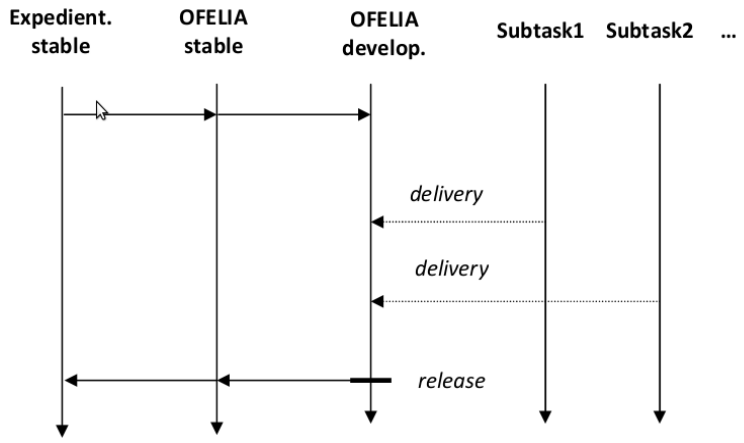


Figure 5: GIT branching schema

3.1.1.4 **SDK**

During the early stages of T5.2 a Software Development Kit or SDK has been developed by T5.2 leaders in order to facilitate and simplify the process of development over the Expedient and Opt-in manager components.

The SDK is a set of two virtual machines (currently VirtualBox images), as follows:

Mininet machine

- File: OpenflowVM.mininet.tar
- Type: VirtualBox image
- OS: Ubuntu 10.04 LTS
- Networking:
  - eth0: uses DHCP
  - eth0:1 : static IP alias: 192.168.254.148/24
- Tools: Mininet and NOX

This VM is supposed to be used for deploying the virtual OpenFlow networks for testing purposes. By means of the Mininet CLI [8] it is possible to create virtual networks of custom topologies. The

[2] Internal development

static IP for this machine is not strictly necessary because the connection with the FlowVisor is realized from the Mininet to the FlowVisor when the network controller is specified See [10].

On the other hand there is NOX version 0.8.0 installed in order to provide a deployable OpenFlow controller.. Nevertheless, for starting a NOX instance it is enough to go to `/home/openflow/noxcore/build/src` and type

```
>./nox_core –i ptcp:2525 packetdump.
```

Expedient machine

- File: OFELIASDK.tar
- Type: VirtualBox image
- OS: Debian Squeeze 6.0
- Networking:
  - eth0: uses DHCP
  - eth0:1 : static IP alias: 192.168.254.193/24
- Tools:
  - GIT code repository of Expedient and Opt-in Manager (apache properly configured)
  - phpMyAdmin
  - FlowVisor
  - OpenFlow reference implementation
  - Editors/ IDEs: Eclipse+pydev , VIM, Emecs

The Expedient machine contains fundamentally the code of the control framework. The web server, in this case Apache2, is configured to use the files contained under the /home/user/ofelia-git hierarchy, which means that development can be directly carried out over the code without having to either install any software dependency nor configuring anything else.

In addition, if one wants to switch from a branch to another issuing the command “git checkout origin <branchname>”, will automatically download the branch code and the web server and the rest of the tools will instantly be working.

The repository files are located at /home/user/ofelia-git/. At this location code is organized as depicted in the following image:

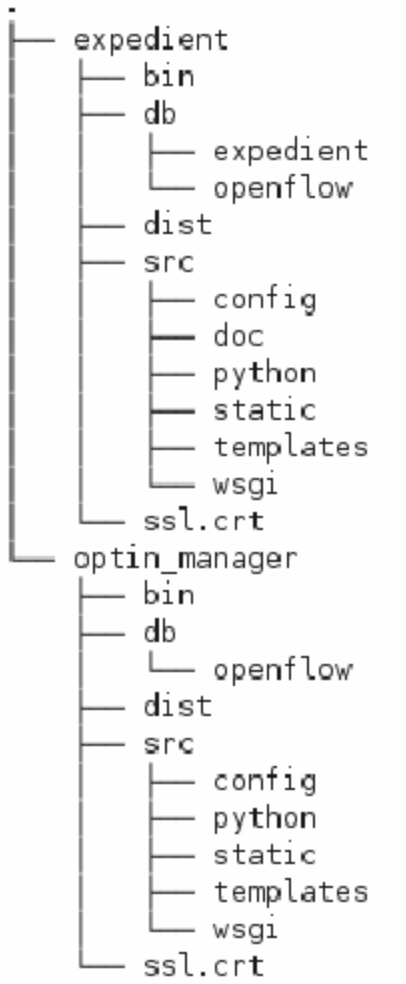


Figure 6: Code directory structure (expedient.stable branch)

3.2 Modified architecture

OFELIA extends Expedient's core functionality with two new modules: a Virtualization AM and an LDAP AM. The Virtualization AM enables users to allocate and control virtual machines on physical servers. The Virtualization software is composed basically of three modules (Virtualization Plug-in, Virtualization Aggregate Manager and the Agent). The agent actually provides an interface to the management interface for managing virtual machines on the physical host and binds this management interface to the aggregation manager. The LDAP module will manage all the issues related to user authentication. Further design details for both modules can be found in Sections 3.3.1 and 3.3.2, respectively.

By maintaining and adapting the Opt-in Manager to OFELIA facility requirements, the user will also be able to set up an OpenFlow test bed. The outcome is a test bed that allows sending flows from one virtual machine to another through an OpenFlow network. The resulting scheme is the one depicted in Figure 7.

The main two additions for the new Expedient architecture are the Virtualization Aggregate Manager and the LDAP module. The Virtualization Aggregate Manager module will enable the Expedient users to allocate virtual machines in Virtual Servers and then control them. The Virtualization Aggregate Manager is composed basically of three modules 1) virtualization Plug-in,

2) Virtualization Aggregate Manager and 3) the Agent and it will be explained in further detail in section 3.3. The LDAP module will manage all the issues related to user authentication.

By maintaining and adapting the Opt-in Manager to OFELIA facility requirements, the user will also be able to set up an OpenFlow testbed. The outcome is a testbed that allows sending flows from one virtual machine to another through an OpenFlow network. The resulting scheme is the one depicted in Figure 7.

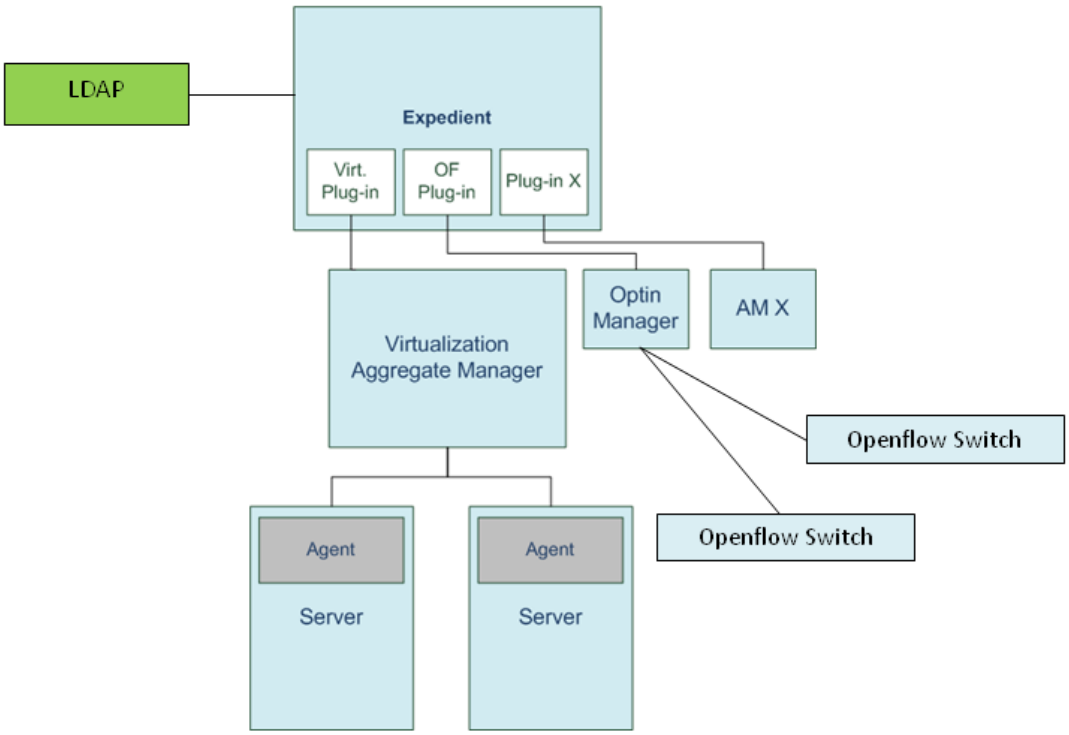


Figure 7: Expedient architecture with the additions.

3.3 Adaptation, expansion and debugging. Sub-task descriptions.

In this section a detailed explanation is shown for all the current coding tasks that under development under T5.2

3.3.1 LDAP integration subtask

The main goal of this subtask is to adapt the current implementation of the control framework to support LDAP-based authentication, in contrast to the current authentication system (authentication against MySQL database).

The authentication subsystem will be used for controlling access to different services in the OFELIA facility including shell access to virtual machines via secure-shell and VPN endpoints for terminating secure connections between an OFELIA slice and the researcher's own lab environment.

All user credentials will be stored in a centralized per-island database and will be accessible via an LDAP compliant interface. Credential information may consist of either username/password pair or public key information.

The software architecture envisaged and tested for this purpose is depicted in Figure 8. The Pluggable Authentication Modules (PAM) framework is used for authenticating by means of user/password pairs. A patched SSH daemon (see **[24]** for details) is also capable to retrieve public keys directly from the LDAP server, bypassing the PAM framework.

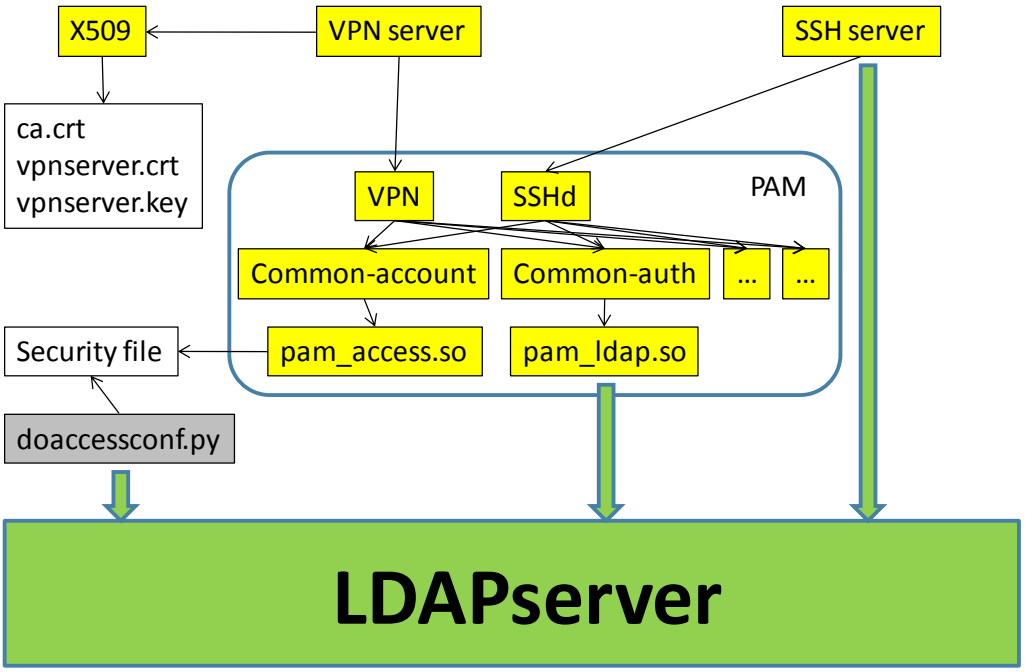


Figure 8: Authentication architecture

The PAM framework **[25]** is the heart of the architecture depicted in Figure 8. PAM allows distinct configurations for each service actually using the PAM framework. For both services the PAM library must be configured for actually accessing the LDAP PAM backend for retrieving all username/password credentials from the LDAP server.

Within OFELIA, many slices and users will use the same physical infrastructure in parallel. VPN endpoints and access to virtual machines must be limited to those entities that are part of the user's slice. The PAM framework provides user identification, authentication and also authorization functionality.

In OFELIA PAM is used both for conducting the authentication and authorization step: the pam_ldap.so module directly verifies the user's credentials by querying the island's LDAP server; the pam_access.so module defines relevant groups (via Unix netgroups) containing users allowed to access resources of a specific slice and all resources assigned to this slice (hosts, VPN endpoints, etc.). An implementation detail: the pam_access.so configuration is stored in a local access.conf file containing all rules for specific slices. A Python script has been developed for retrieving the relevant group information from the LDAP server and updating this configuration file accordingly. (if needed, the script can be repeatedly executed as part of a CRON job for example). The netgroup information in the LDAP server contains: netgroups of users, netgroups of machines and a definition of which of the former netgroups are granted access to which of the latter netgroups. By manipulating the netgroups in the LDAP directory, users or resources may be assigned dynamically at run-time to an existing slice.

Besides the authentication and authorization of the user through user/password combinations, authentication based on certificates or public keys is possible. The OpenVPN service can make use of X.509 certificates. It only needs to store its Certificate Authority (CA) certificate and its own certificate (signed by that CA) and key files. In other words, it does not need to store per user

certificates: when logging in, it checks whether the certificate presented by the user was signed by the CA and, in case it is, it is assumed that the user is entitled to setup the OpenVPN connection. By applying the lpk patch to the OpenSSH daemon, we are also able to store public keys for secure shell based access in the LDAP directory, thus allowing ssh based login by using the corresponding private key, without the need to store the public key on the machine itself. The centralized LDAP based user credential storage simplifies user management considerably. Enabling non username/password pair credential authentication methods is key for automated and/or batch mode login without needing to (re)enter or cache the username/password combination.

It was also decided that beside the master LDAP server in the hub island, other islands will replicate this LDAP server in order to improve the reliability and performance. From the documentation, it became obvious that several modes exists to deploy replica, proxy or whatever LDAP server configurations. However, in the end no reliable configuration can be realized where write operations can be performed on the local LDAP (replica) server and later on synchronized in both ways with the master central LDAP server. Therefore, as depicted in Figure 9, write operations (typically rarer than read operations) shall still happen on the central LDAP server which then gets replicated through sync-replication processes into the local LDAP servers, from which then local read operations are performed. The sync-replication processes can be either push or pull mode operations: the pull mode of operation will be preferred as it keeps the replicas as up to date as possible.

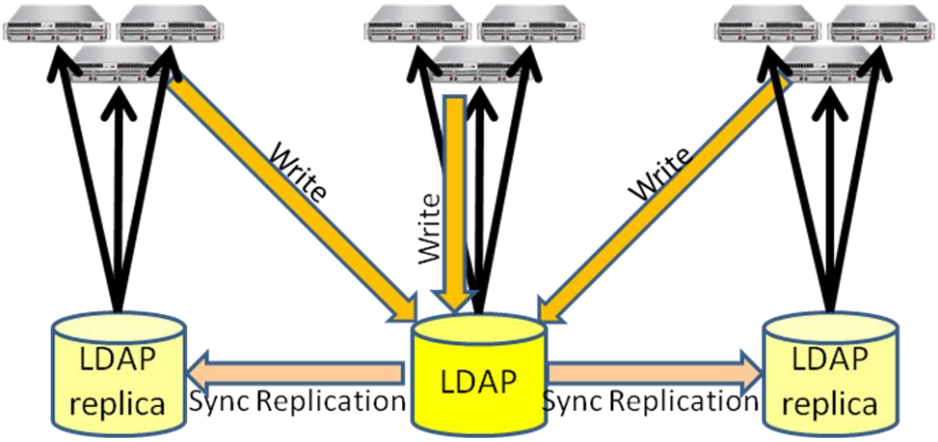


Figure 9: Replication of LDAP server

For this purpose, the Django module to authenticate against LDAP has been used: http://packages.python.org/django-auth-ldap/, modifying the source code of Expedient tool in order to use the LDAP authentication prior to the MySQL engine.

Please note that the MySQL backend is not replaced, rather the LDAP and MySQL backends are used in parallel for storing the relevant user credentials. Though, this solution is not optimal, it is still feasible for OFELIA. When multiple backends are used in parallel, they have to be ordered for prioritization. In the case of the control framework, the implementation carried out in the ldap branch of the GIT repository, has taken the following approach:

1. LDAP authentication Django backend
2. Basic HTTPs authentication against MySQL Django backend
3. Web-based authentication against MySQL Django backend

3.3.2 Server virtualization software subtask

3.3.2.1 Objectives in short and long term

One of the fundamental requirement for the OFELIA control framework was the capability to manage virtual machine (VM) instances that can act as endpoints, controllers or any other required function inside an experiment. During the research period at the beginning of the first phase in the OFELIA project, many applications intended to carry out this actions, were studied[1]. Nevertheless, sometimes because of lack of capabilities, different goals, difficulty of adaption or other reasons, it was decided to create a new specific tool to deal with virtualization requirements in the control framework.

The development of the application and its subcomponents is planned considering requirements for short and long term. The idea is to have a first stable functional module capable of doing the basic although not definitive functions and extend it to the final model gradually during phase 2.

For the first release a basic set of requirements were identified:

- Limiting the virtualization technology supported to Xen in order to focus in the requirements regardless of the underneath hypervisor.
- Regarding the definition of the virtual machines, a basic template will be stored in the same virtualization server.
- Because of simplicity a unique VM file-disk image template with a Debian Lenny distribution is considered. It will contain all the applications identified in Milestone 5.1 as basic for experiments (FlowVisor, NOX, Wireshark, OpenFlow, etc).
- In a first development users will be registered and authenticated against the application database. Nevertheless, in further development path a centralized LDAP will be used to improve user management in the whole facility.
- The user will be able to realize four kinds of actions against the server: create, start, stop and delete VMs through the plug-in interface in the Expedient framework.
- In order to ensure the VMs connectivity and avoid duplicate addresses, an IP and MAC addresses manager module is required. Its goal is to define the address space defined by project, slice and the aggregate manager, which will handle the specific VMs.
- Inside the server it is important to ensure slice traffic isolation and prevent IP/MAC spoofing.

For the second phase many improvements are identified:

- Virtualization technologies support should be extended. OpenVZ, VirtualBox, etc. may be reconsidered.
- Regarding the VM template storage it is expected to have a remote template repository, from which the required VM file will be downloaded to the virtualization server.
- The number of different templates available will be increased.
- Users will be allowed to upload their own VMs to the facility repository.
- Instead of using file-disks, because of efficiency reasons, the VMs will be created over a Logical Volume Manager (LVM).
- The set of actions over a VM will be increased adding the possibility of changing the characteristics (RAM memory, disk image, etc) of existing VMs once created.
- Server monitoring will be added.

- The communication will be completely SFA compliant. . However, adopting the SFA framework is for further study.
- A policy manager module will be implemented in order to allow each Island Manager (IM) to decide a set of rules the actions against his servers should fulfill.

3.3.2.2 Architecture

Starting from the standard architecture followed by Expedient and Opt-in Manager, the Virtualization Technology Plug-in is composed by three modules as depicted in Figure 10:

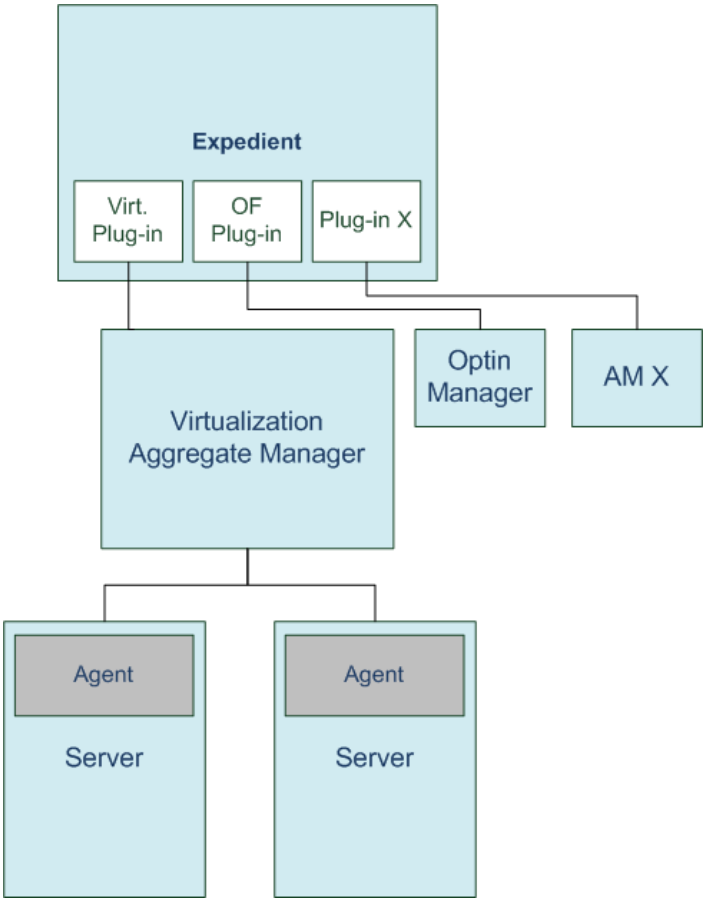


Figure 10: General Architecture considering the plug-in-AM-Agent triplet

- **Virtualization Plug-in in Expedient**: Since Expedient acts as a plug-in holder, the specific plug-in should be placed in it. It is responsible to add to Expedient the required functions and interface to allow the user interaction and the communication with the Virtualization Aggregate Manager (AM) through its southbound communication layer. Since the Plug-in takes advantage of the Expedient functionality (i.e. it uses Expedient database, Expedient's URL dispatcher, etc), it does not have a well conformed architecture, but it uses Expedient one and increase its capabilities (specific XML parser/crafter, communication with VT AM functions, etc).
- **Virtualization Technology Aggregate Manager** (VT AM): It is the Opt-in Manager equivalent in the virtualization part of the facility. A block diagram of it is depicted in Figure 11. As an AM it manages all the resources assigned to him. In this case the resources are the virtualization servers underneath. It takes control of the actions against the server and communicates them to the Agent.

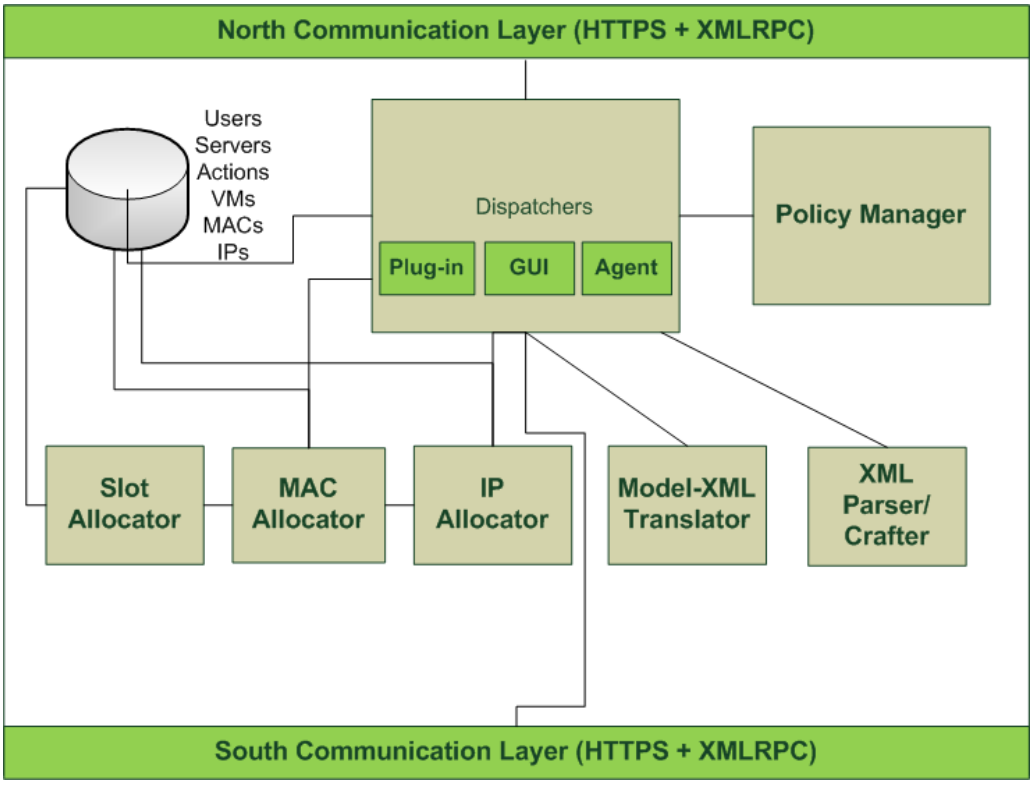


Figure 11: VT AM block diagram

- Southbound and northbound communication layers through HTTPS and XMLRPC
- Dispatchers: In this module there are concentrated the three available dispatchers. The first one is the one attending the queries that come from the plug-in through the northbound interface. The second one handles those queries that come from the Graphical User Interface (GUI), meaning the actions done by the IM through the website. The last dispatcher is the one which receives the responses of the queries done against the server Agent. The communication model will explained in more detail in the following section.
- Policy Manager: Will apply the local AM policies over the received requests from the plug-in. At the moment it is implemented as a dummy block.
- XML Parser/Crafter: Parses and crafts from/to the XML string/intermediate class representing the XML structure.
- Model-XML Translator: Translates from the intermediate class that represents the XML structure to the data model classes of the application, and in the inverse direction.
- Slot Allocator: When creating a VM for a specific project and slice, it sets a slot based on the AM, the project and the slice for later providing a MAC address.
- MAC Allocator: Using a certain slot generated by the Slot Allocator, completes the MAC address string to generate a proper host MAC address.
- IP Allocator: Manages IP addresses to provide the control interfaces when a VM is created.

- **Server Agent**: It is the responsible of acting in the server and managing the hypervisor installed in it. It receives orders coming from the VT AM in its north communication layer and configures and executes the Xen scripts to accomplish the user petition.

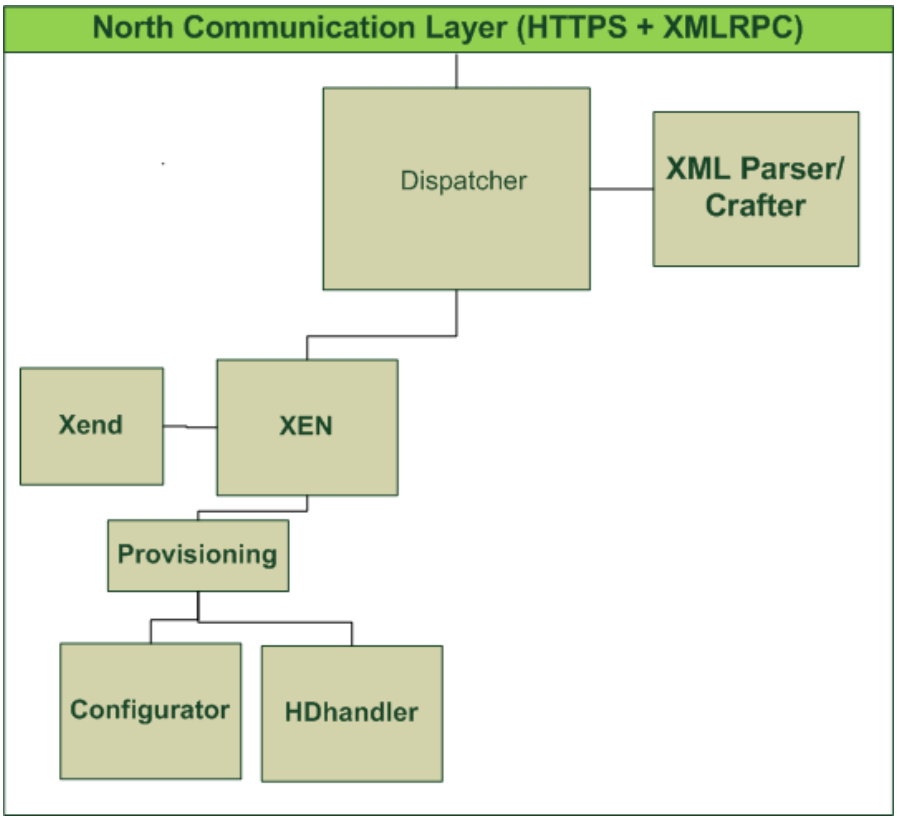


Figure 12: Server Agent block diagram

  - Northbound communication layer receives the queries coming from the AM
  - Dispatcher: Selects the corresponding dispatcher depending on the virtualization technology. At the moment only Xen is supported.
  - XML Parser/Crafter, same as in the VT AM.
  - XEN: Xen dispatcher
  - Provisioning: Handler of the Provisioning type actions (create, delete, start, stop).
  - Configurator: Generates the configuration file to create the required VM.
  - HDhanler: Manages the copies of the VM disk-image file templates.
  - Xend: Executes proper Xen commands.

3.3.2.3 Utilized Tools

This section describes the set of tools utilized in the development. The biggest constraint came from the fact of using Expedient and Opt-in manager as starting points, resulting in the use of Python as the programming language. All the development was done in the dedicated SDK VM.

- Programming language : Python 2.6
- Django Web Framework [11]
- Xmlrpclib [12] and rpc4django [13] : used as communication modules.

- generateDS [14]: generates Python data structures (for example, class definitions) from an XML Schema document. It also generates parsers that load an XML document into those data structures.
- GIT code repository for version control and code sharing.
- VirtualBox [15]: used to virtualize the SDK VM.
- Xen: hypervisor running in the servers.
- OpenVSwitch [16]: virtual switch placed between the xen Dom0 interface and the different DomU (VMs) virtual interfaces in order to isolate each VM's traffic and prevent spoofing.
- Eclipse with Python plug-in and Vim for editing

3.3.2.4 **Code structure**

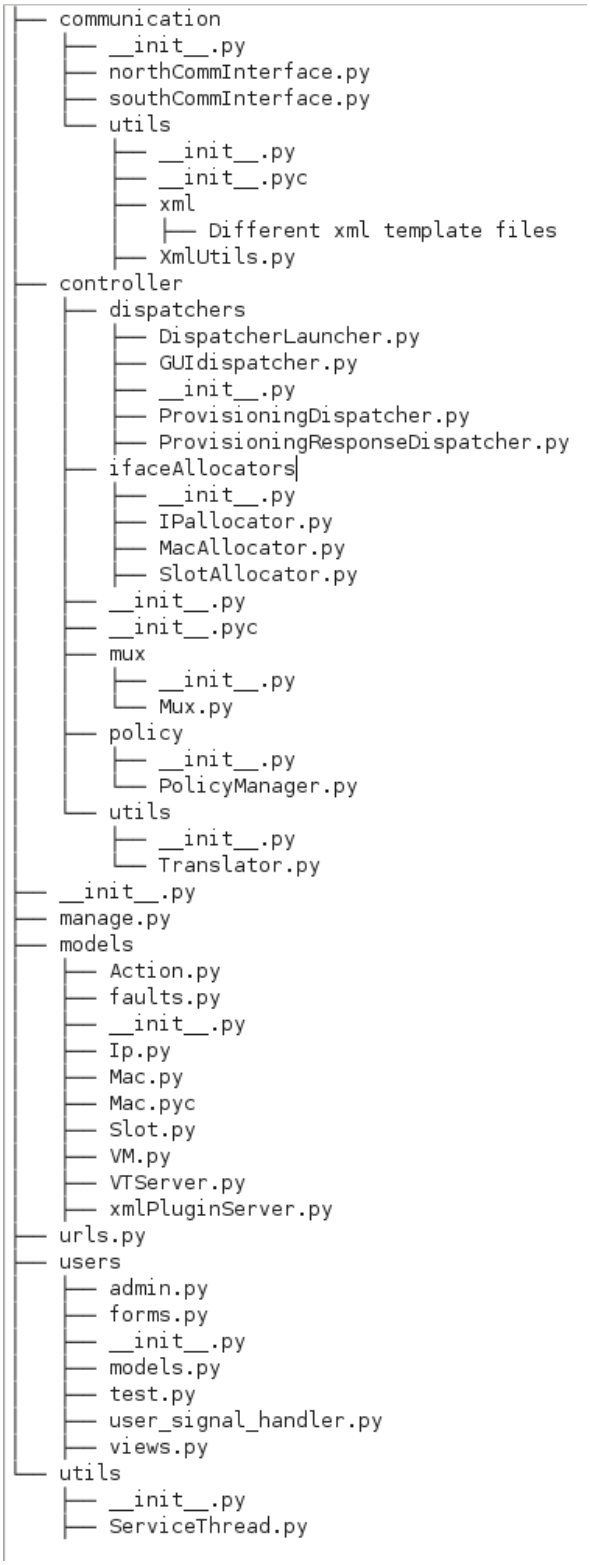
In this section the structure of the code of the different modules will be detailed. Based on the directory tree, the location of each component is explained.

- Plug-in: ofelia-git/expedient/src/python/vt_plugin

```
├── communication
│   ├── __init__.py
│   ├── southCommInterface.py
├── controller
│   ├── __init__.py
│   ├── __init__.pyc
│   ├── ProvisioningDispatcher.py
│   ├── ProvisioningResponseDispatcher.py
├── forms.py
├── __init__.py
├── log.txt
├── models
│   ├── Action.py
│   ├── Action.pyc
│   ├── __init__.py
│   ├── Mac.py
│   ├── VM.py
│   ├── VM.pyc
│   ├── VtPlugin.py
│   ├── VTServer.py
│   ├── xmlrpcServerProxy.py
├── templates
│   └── vt_plugin
│       ├── aggregate_add_virtualmachines.html
│       ├── aggregate_crud.html
│       └── vt_plugin_base.html
├── urls.py
├── utils
│   ├── __init__.py
│   ├── __init__.pyc
│   ├── ServiceThread.py
│   ├── Translator.py
├── views.py
```

- communication: southbound interface and its method implementation.
- controller: queries and response dispatchers
- models: data models required for the application. All of them inherit from the django *models.Model* class in order to store their instances in the Expedient database.
- templates: html templates for the webpages.
- urls.py : application web addresses and corresponding function definition.
- utils: different utilities. At the moment, thread dispatcher and translator module (from the parsed XML classes and the plug-in data models).

- Moreover, some additional functionality has been implemented in the file /ofelia-git/expedient/src/python/ui/html/views.py. These functions work as a kind of user interface dispatcher receiving the queries from the user interaction with the web site and then calling the "provisioning dispatcher" and redirecting to the next web page. This code will probably be reallocated in a more adequate place.
- VT AM: `ofelia-git/vt_manager/src/python/vt_manager`



- communication: Implementation of communication methods in each interface. In utils/ there is XmlUtils (containing the XmlHelper class which implements all the parsing/crafting actions and xml generators). In utils/xml the xml template files are present

- dispatchers: Query, response and GUI dispatchers.

- ifaceAllocators: IP and MAC address managers

- mux: not implemented yet.
- policy: Policy Manager. Dummy class at the moment

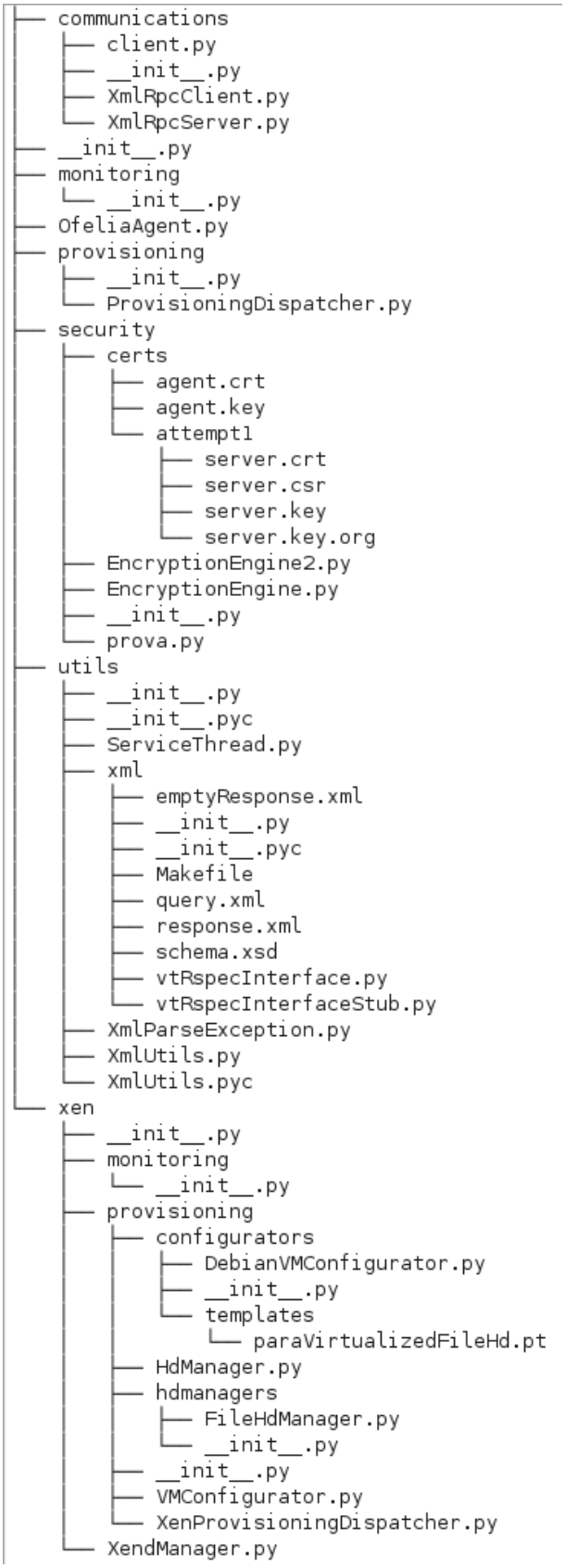
- utils: Utilities. Translator class translates from/to xml structure class/data model

- models: data models required for the application. All of them inherit from the django *models.Model* class in order to store their instances in the VT AM database.
- urls.py: application web addresses and corresponding function definition.

- users: users administration. Just basic functions. Original code copied from Opt-in manager. User administration will probably be changed to a custom model.

- utils: Different utilities.

Agent: `ofelia-git/vt_manager/src/python/agent`



- communication: XMLRPC client and server.
- monitoring: not implemented yet.
- OfeliaAgent.py: python executable script to run the Agent as a daemon in the server.
- provisioning: dispatcher of the provisioning type queries.
- security: not implemented yet.
- utils: xml template files and parser/crafter. Class generators from xml files.
- xen: Xen specific dispatcher
- monitoring: not implemented yet.
- VMConfigurator.py & configurators: VM configurators. They generate VM configuration file.
- HdManager.py & hdmanagers: At the moment, only the FileHdManager is implemented, which performs file-disk image's copy operations.
- XendManager.py: Executes Xen commands in the server.

3.3.2.5 Communication model between modules

The communication model implemented for the three modules 1) Expedient's plug-in 2) AM 3) Agent is based on XML-RPC. Each of the three components offers the corresponding methods to carry out the needed functionalities.

The communication process is done by two methods as depicted in Figure 13. The north interfaces publish the send( ) method and the south interfaces the sendAsync( ) one. The model works as follows:

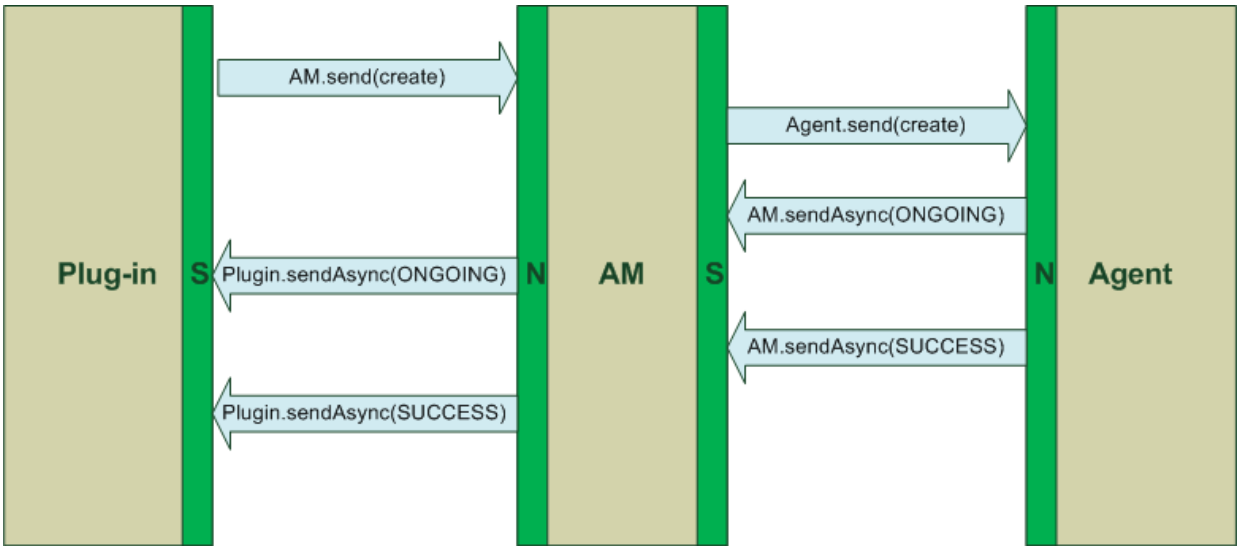


Figure 13: Communication model between the three modules

- When the Plug-in module generates the XML file (RSpec) containing the actions that need to be done, it sends the XML as a string using the `send()` method published by the north interface of the AM module underneath.
- When the AM receives an XML string through its `send()` method. It parses the string and processes the query doing what is required. Then it regenerates the XML string with the new information if there is any and sends it to the Agent by using the `send()` method on the Agent module's north interface.
- Then the Agent receives an XML, if the parsing of it is successful automatically sends an update with an ONGOING status for each of the actions being performed. This is done through the `sendAsync()` method in the south interface of the AM.
- The reason why this first ONGOING status update is sent is to notify the AM that the XML has been received correctly and, since some actions (i.e., create a VM implies copying whole disk images) may require a long time to be completed, this way the AM can continue doing its work (non-blocking operations).
- At the moment that the AM receives the ONGOING status update from the Agent, it resends this info to the Plug-in because of the reasons explained in the previous point.
- When the Agent finishes a query it asynchronously sends a status update SUCCESS/FAILED of the action to the AM, which in addition to perform the required actions, resends this update to the Plug-in finishing the cycle.
- The connection between modules is done over HTTPS and uses basic user:password authentication.

The structure of the XML used for the communication up to now is the following: The first classification is query/response XML. It is sent with `send()` method and the second with `sendAsync()`.

Secondly, the actions super-type can be provisioning or monitoring. Until now, only provisioning has been implemented. Within the provisioning tag, the action itself (create, delete, start, stop) is defined. It has an id which is actually an UUID so that the AM and Plug-in can recover it when an update is received through `sendAsync()`.

Between the <action> tags, the VM is defined. There can be only one VM per action, but many actions per provisioning in each query.

3.3.3 Adding support for ProtoGENI-enabled equipment subtask

The IBBT Virtual Wall is controlled by the EmuLab software. This software is a control framework where experimenters can create their experiments in terms of network topology and what images to launch on each of the network nodes. Recently the developers of the EmuLab software have been integrating an EmuLab specific SFA interface called ProtoGENI [7]. As Expedient has been selected as control framework for the OFELIA facility and SFA is targeted as federation architecture, the plan is to develop a ProtoGENI plugin that will enable Expedient to interact with the EmuLab software.

Goal:

As shown in Figure 14 the EmuLab software controlling the IBBT virtual wall is a complete control framework on its own. It has both a web-based user interface and an XML-RPC API through which users can control and manage their experiments. Recently, an additional user interface module has been added that implements the SFA Aggregate Manager (AM) API. Specific to the ProtoGENI API is the EmuLab specific RSpec: already a 2nd version of this RSpec has been released. At the other side, the EmuLab software coordinates and executes the proper commands to launch the actual experiments on the infrastructure, implying that the proper machines are rebooted with the selected image and that these nodes are interconnected by a central switch in the topology as requested by the user.
At the other side, the Expedient framework, selected as the base of the OFELIA control framework, features the Opt-in manager plug-in and a PlanetLab plug-in in addition to the similar user interface modules (web-based, GENI API (xml-rpc), …). The Opt-in manager and PlanetLab plug-ins talk to the corresponding aggregate managers on the infrastructure side. The PlanetLab plug-in is an SFA-like plug-in developed for communicating with a PlanetLab AM. Despite a ProtoGENI plug-in, which was announced by the Expedient owners, these plans have not been materialized and are canceled. Therefore, the intention of the OFELIA project is to develop such ProtoGENI plug-in.

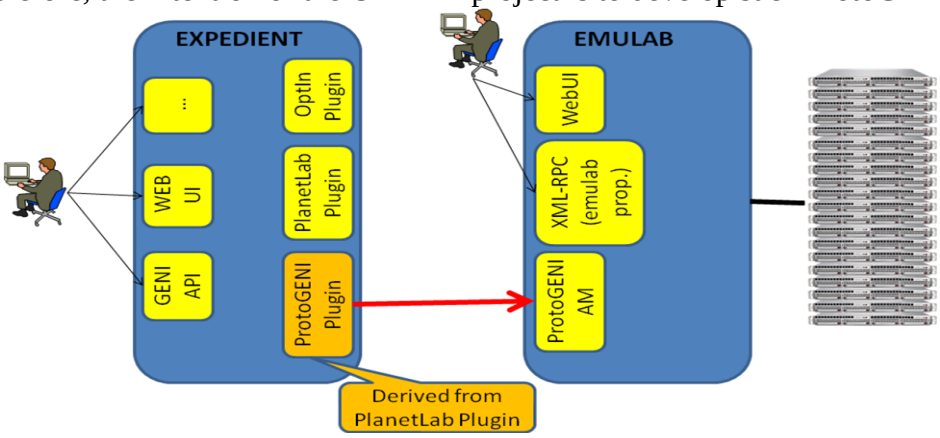


Figure 14: Position of ProtoGENI plug-in in the Expedient control framework

Given the fact that the PlanetLab Plug-in is implementing the SFA AM interface (although specific for PlanetLab) and the EmuLab ProtoGENI interface is also an SFA AM API, we came to the conclusion that we ideally should try to build a ProtoGENI plug-in in Expedient, starting from the PlanetLab plug-in.

The main difference would be that the ProtoGENI specific RSpec will require its own handling rather than handling the PlanetLab RSpecs. However, from a very high-level viewpoint, conceptually both RSpec types represent a network meaning a collection of nodes and links rather than FlowSpace specifications in OpenFlow (recall that the purpose is in the end to allocate a network of nodes, on which some nodes a software OpenFlow switch will be launched that will connect to another node on which an OpenFlow controller is launched- rather than obtaining a slice of an existing OpenFlow network).

3.3.4 Adapting optical equipment to OpenFlow

3.3.4.1 Related backgrounds

The underlying principle of OpenFlow is to treat traffic as flows, either packet-based or circuit-based traffic at different granularity. The idea behind OpenFlow is to have the control functionality taken out of the equipments (i.e., Switch, Router, or an ROADM) to a centrally managed or a distributed system, while retaining only data plane on the equipment. As the data plane is implemented in the hardware, OpenFlow provides the control plane with a common hardware abstraction. A network is managed by a network-wide operating system running on top of a controller that controls the data plane with OpenFlow protocol. The OpenFlow controller is a server, which has the capabilities to host different network management and control applications to effectively manage the network in a centralised or distributed way. This separation between the control and data plane and the capability to treat packet and circuit traffic as flows makes OpenFlow protocol a single standardized control for both packet and circuit networks. There have been several attempts and proposals to control both circuit switched and packet switched networks using the OpenFlow protocol. Figure 15 shows a unified architecture OpenFlow provides.

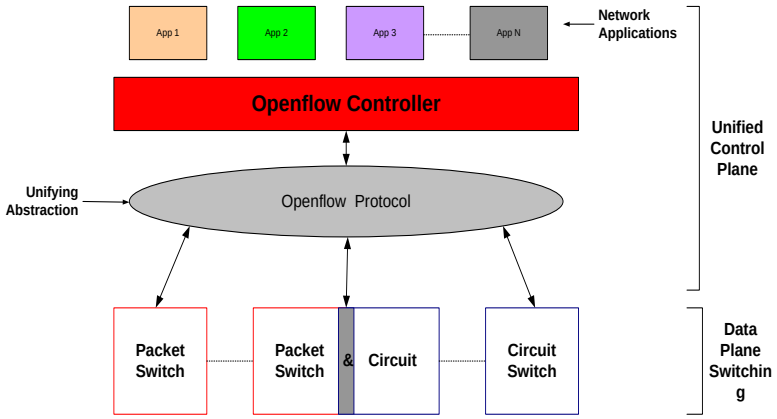


Figure 15: OpenFlow unified architecture

OpenFlow abstracts each data plane switch as a flow-table. The control plane makes decisions as to how each flow is forwarded (reactively as new flows are detected, or proactively in advance), then caches its decision in the data plane's flow-table. For instance, the control plane might decide to route all of the http traffic destined to Europe along the same path, and so would add a flow-entry in the flow-table of each switch along the path. OpenFlow provides various types of actions on flows (e.g., forward, multicast, drop, tunnel, etc.) as outlined in the current OpenFlow protocol specification. A flow can be defined in a flexible way as a combination of any L2, L3, L4 headers

of a packet or L1/L0 fields of a circuit flow. For L2/L3/L4, any combination of the 10 tuple header (Figure 16) can be used to define a flow. Example: All traffic destined to a particular Destination IP address 10.10.0.10 can be defined as a flow. Another flow can be all HTTP (TCP port 80) traffic. Similarly L1/L0 can also be treated as flows and it can be a combination of the field shown in Figure 16.

In Port	VLAN ID	Ethernet			IP			TCP	
		SA	DA	Type	SA	DA	Proto	Src	Dst

Figure 16: OpenFlow switch flow table entry

In/Out Port	In/ Out Lambda	VCG	Starting Time-Slot	Signal Type

Figure 17: Circuit-based flow table entry

Incoming packets are matched against the flow definitions; if there is a match, a set of actions are performed, and statistics will be updated. Packets that do not match any flow-table entry are (typically) encapsulated and sent to the controller. The controller can decide how to process the packet and then (optically) insert its decision in the data plane (i.e., a new rule in the flow-table) so consequent packets in the flow are processed the same way, without contacting the controller. Therefore, while each packet is switched individually, the flow is the basic unit of manipulation within the switch. Note that by switch we are refereeing to a generic OpenFlow enabled switch. The fields of the flow table are shown in Figure 18. The rule field defines the field from the 10 tuple header to be matched in the packet. Action defines the way the packet should be treated depending on the rule. Statistics field is used to gather packet statistics which are used by the network applications to make dynamic and automatic decisions.

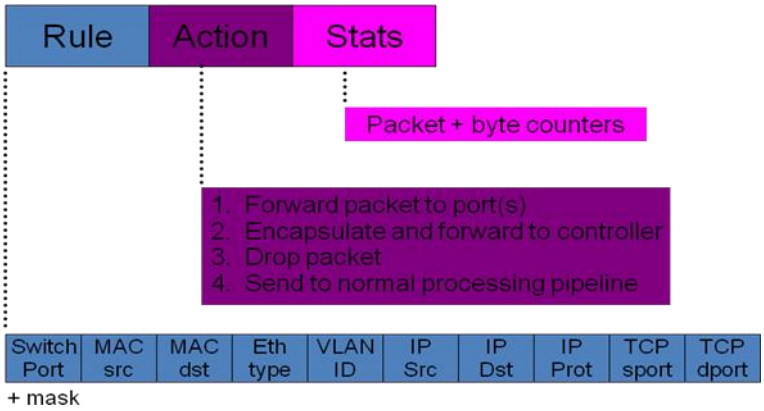


Figure 18: Flow table structure

Authors in [17] showed that cross-connect tables in transport network elements can also be considered as OpenFlow flow tables. Flow can be defined as circuit flows using layer 1 time-slot switching based on SONET/SDH and virtual concatenation (VCGs) or L0 wavelength or fibber switching (see Figure 17). Therefore, the OpenFlow architecture allows for 1) a flexible definition of what constitutes a flow, at what layer and switching granularity and 2) the definition can be changed dynamically, over time or in different parts of the network; for instance to aggregate flows as they move from the edge to the core of the Internet. For a circuit switch, cross connect entries could be made using the fields in the circuit switch's flow table as follows:

In Port	In Lambda	VCG	Starting Time-Slot	Signal Type

⇔

Out Port	In Lambda	VCG	Starting Time-Slot	Signal Type

In contrary to packet networks, the circuit switch flow table is not used to look up flows, as circuit ports do not have the visibility to the packets. There is no buffering in the circuit switch and no packets are forwarded to the controller. The controller is responsible for pre-provisioning or tearing down the connection of the circuit flows. Dynamic provisioning of the circuit is done by user-defined applications written on the OpenFlow controller. In a simplified case of optical switch, a flow table is a database inside the switch controller OS, which is holding cross connections status between incoming port/incoming wavelength and outgoing port/outgoing wavelength.

There are different versions of OpenFlow controllers currently available. NOX controller is one of them, which runs on Linux (Debian/Ubuntu). NOX controller is written in C/C++ and provides a platform for writing network control and management applications in Python or C++. SNAC is another version of OpenFlow controller, which has a more sophisticated GUI to manage the network and gather statistics.
A key component of the OpenFlow architecture is the flow level virtualization of the network and its resources. The programmability ingredients of virtualization are provided by the OpenFlow API, where clients can program the switches by flexibly defining flow according to their needs and inserting them into the flow tables. The isolation ingredient of virtualization is provided by virtualizing the API itself with a thin layer of software, which is called FlowVisor [19]. The FlowVisor is housed outside the switch leaving both the data plane and the controllers un-touched. The FlowVisor is transparent both to the switched and to the controllers and it enforces traffic isolation by monitoring and re-writing OpenFlow protocol messages. Therefore, the switches think that they are talking to a single controller, while each controller thinks that it is controlling its own set of OpenFlow enabled switches. OpenFlow enabled virtualization (i.e., FlowVisor) allows the transport service provides (e.g., network operators) to retain control over the transport network, while allowing clients (such as an ISP) to use whatever automated intelligent control algorithms they may desire in their isolated slice of the network as depicted in Figure 19.

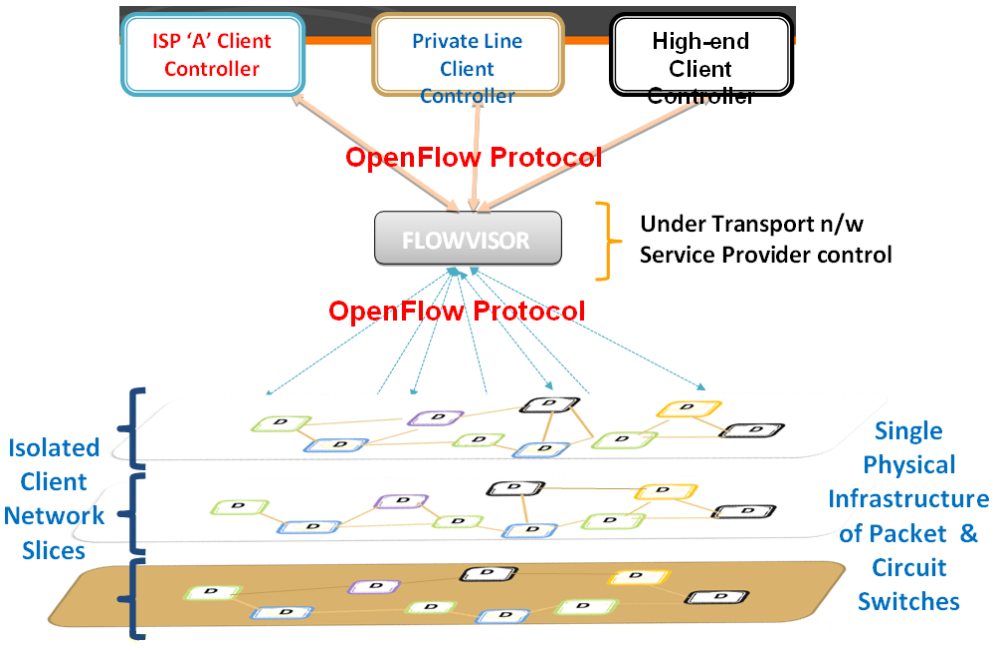


Figure 19: OpenFlow virtualization in Circuit and Packet switching networks

The transport network resources provided to the ISP by the transport service provider can further be virtualized by the ISP for its own needs. This means that we can further virtualize the client network via a FlowVisor, which is under the control of the ISP.

3.3.4.2 Packet to circuit mapping

This section contains a few examples showing one of the ways of mapping packet to circuit traffic using OpenFlow.

Ethernet to WDM flow mapping: The switch registers with the OpenFlow controller and exchanges its features (e.g. OpenFlow version, flow table size, etc) with the controller. The OF controller with a user-defined rule for the flows decides on how the packet should be forwarded. The OF controller pushes the rule into the flow table of the switch. A cross connection is made based on the rule and the traffic flow is switched. For instance, a flow can be defined as all traffic destined to a specific network 192.168.1.0/24 which belongs to VLAN 20 and is mapped to a particular wavelength λ1. A new flow can be added to the same wavelength λ1 or it can be added to a new wavelength λ2.

wport = 1, wavelength = λ1 maps to wport = 3, wavelength = λ1

A new flow can be added to the same wavelength or it can be encapsulated into a new wavelength and switched out of a different port.

Example of Ethernet to TDM flow mapping: The switch registers with the OF controller (NOX) and exchanges its features with the controller. NOX Controller pre-provisions the SONET/SDH Virtual Concatenation Groups (VCG). It then maps the L2/L3/L4 fields to VCGs and VGCs to the port, time signal and start time. This rule is pushed to the flow table by the OF controller. VCGs acts as a virtual port between packet and circuit switch fabrics. L2/L3/L4 fields based on the flow definition can be mapped onto the respective VCG. In case of a TDM switch, if the Link Capacity Adjustment system is turned on then an application can be written on the controller for dynamic provisioning of the bandwidth. For Example, All VLAN 10 traffic can be grouped into VCG10. VCG10 can be mapped onto the physical ports with the set of STS-n and starting time slots.

On the other end, the traffic received from VCG is switched to the ports based on the VLAN tag. By writing different network applications on top of the OpenFlow controller, the control and provisioning of bandwidth of the network can be dynamic and automated. In the next section, the possible approaches to integrate packet and circuit domain using OpenFlow is discussed.

3.3.4.3 Interlayer Open Flow operations

A very important aspect of OpenFlow controller for Layer1 / Layer 0 architecture is how it can be integrated with OpenFlow controllers for upper layers in particular Layer 2. One of the possible approaches is depicted on Figure 6. For more clarity the OpenFlow controllers for Layer 2 and Layer 1/0 are shown as separate entities but in practice they reside on the same box as a single entity.

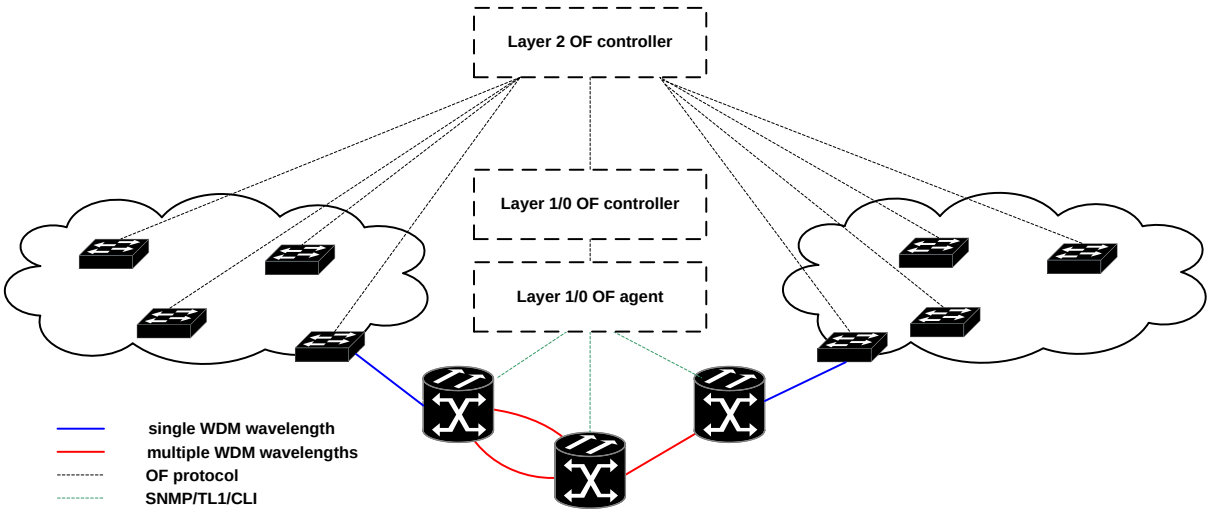


Figure 20: Multi layer L2-L1/0-L2 islands

Ethernet switches form a kind of Layer 2 islands that are controlled by a single controller. The picture doesn't reflect exact OpenFlow architecture as shown in **Figure 19**, where each Layer 2 device are OpenFlow enabled and between OpenFlow controller and OpenFlow agent there is a FlowVisor [19] functional block, which is responsible for slicing of the underlying physical network infrastructure. The FlowVisor can be safely ignored from this discussion as it is an OpenFlow controller by itself with enhanced features. Another simplification is depicting a single OpenFlow controller blocks. In theory there might be many such objects for each layer. This aspect is quite important from the slicing perspective, further discussed in Section 3.3.4.6. There is one more simplification which is omitting another possible island connected to WDM device at the bottom of the picture.

Layer 2 OpenFlow controller capable of creating flows within the islands, having visibility into each island should also have knowledge when particular flow request needs to cross the Layer 1 / Layer 0 island and trigger appropriate operations on WDM devices through its controller. The request should define endpoints for the connection in Layer 1/ Layer 0 as its controller doesn't have visibility to Layer 2 islands. Management interfaces to optical switches were depicted with green lines to put stress on fact that OpenFlow agent in this approach is not meant to run on WDM devices. It will communicate with them through one of the available interfaces. An open item remains on how the OpenFlow controller and OpenFlow agent should communicate together and if the means are provided by OpenFlow protocol are sufficient for that are discussed in the following sections.

3.3.4.4 Layer1 / Layer 0 slice concept

Within the OFELIA project a concept of Layer 1 / Layer 0 slice parallel to Layer 2 slice should be defined. As WDM and Ethernet are different techniques the slice definition is also different. Within the project different Layer 2 slicing concepts are being discussed (e. g. MAC based, VLAN based). For WDM an intuitive way of slicing is based on resource which is a wavelength. A difference is that a wavelength is a physical resource and we cannot allocate more lambdas than are currently available. The things look different in Ethernet. For the flows we may allocate abstract bandwidth. Summarized allocated number may be higher than real capacity of the link thanks to statistical multiplexing of the flows. Two approaches may be considered:

- a slice defined as a fixed set of wavelengths,
- a slice defined as a number of wavelengths available from in the whole spectrum.

The latter one is more flexible and in many cases may provide better network resources utilization. Higher flexibility comes as always with higher complexity. To name one, how to guarantee, that the slice 'n' will have 'm' wavelengths available at the each port of virtual switch? The problem is much easier if the set of wavelengths is fixed and reserved for all ports.
Regardless of chosen approach the functionality is embedded in the FlowVisor for Layer 1 / Layer 0. Any request to create a WDM tunnel coming from any Layer 1 / Layer 0 controller should go transparently through that functional block and checked according to implemented policy.
Introduction of the FlowVisor and slicing for Layer 1 / Layer 0 enables operation of multiple OpenFlow controllers managing the same OpenFlow agent. This is expected to work in the same way as for packet-switched OpenFlow. The next section of the document provides a high level design on the integration of Layer 2 and Layer 1/0 using OpenFlow and also explains the building blocks within the OpenFlow controller.

3.3.4.5 High level design

In this section the high-level design of the integration scheme to adopt OpenFlow into the ADVA ROADMs is presented. We assume that ADVA GMPLS protocol stack (or alternatively the SNMP interface to the ADVA ROADMs for configuration) is available. Therefore, we propose an extended OpenFlow controller that is able to communicate with an OpenFlow agent and the GMPLS control plane. The high-level design of this integration scheme is depicted in Figure 21. The Optical Connection Controllers (OCCs) are entities from the optical circuit switching domain, which control the optical nodes (i.e., ADVA ROADMs). The OCCs are connected to the Data Communication Network (DCN) or control plane network for the optical circuit switching domain.
First of all, we have to distinguish two operation domains in this high-level design. The first domain is the packet switching domain and the second one is the optical circuit switching domain. The Add/Drop interface of the ADVA ROADMs is connected to an OpenFlow enabled agent (or for simplicity in implementation to an OpenFlow switch). We assume that the input/output traffic to/from the add/drop ports are encapsulated in Ethernet frames. The DWDM interface of the packet domain is connected to an OpenFlow enabled switch. When the Ethernet frame is received by the OpenFlow enabled agent, it checks its flow table entries. Assuming that the flow table is empty, the OpenFlow enabled switch requests the (extended) OpenFlow controller to provide a proper action for the given packet. The extended OpenFlow controller has a control plane agent, which participate in GMPLS related communications. The Global Flow table is in fact a mapping table between the OpenFlow flows and corresponding lightpaths, which are established in the optical circuit switching domain.

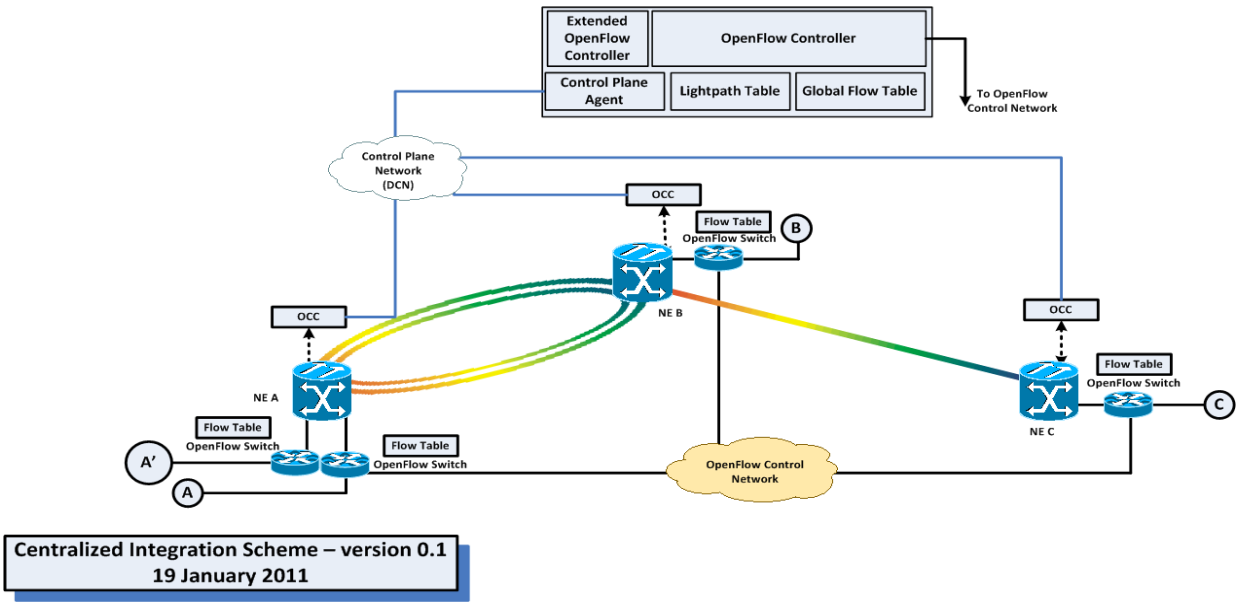


Figure 21: Enabling ADVA ROADMs with OpenFlow

In Figure 21, let's assume that a packet is received on port A of the OpenFlow agent. Since the flow-table is empty (learning phase), the OpenFlow controller (using the higher level application, which is running as a NOX application), computes the proper lightpath for the first frame of the incoming traffic. In order to make this decision, the controller should participate in the communications of a routing protocol (e.g., OSPF-TE). Then the extended OpenFlow controller establishes the proper lightpath for the given traffic. The extended OpenFlow controller utilizes its GMPLS protocol set (or SNMP interface), to establish a lightpath from a certain source to a destination (for example, upon receipt of the first frame). In order to establish a lightpath, we can either use the GMPLS protocol sets of the ADVA ROADMs or we can utilize an SNMP interface to the equipment. In both cases the control plane of the circuit switching domain establishes a lightpath from a source to the destination. The extended OpenFlow controller then updates the flow-table of the intermediate switches as well as its lightpath table and after that any packet from the add/drop port of the ROADM, which matches the specific rule, will be forwarded to its destination port (through the established lightpath). The statistics field in the flow table can be used to keep track of the established lighpaths and in case that the lightpath remains idle for a given amount of time, the extended OpenFlow controller can use its control plane agent to trigger the tear down process for these lightpaths.
In the further sections of the document the scenario where OpenFlow agent uses SNMP interface to GMPLS-based control plane is described.

3.3.4.6 OpenFlow Optical Components:

In this section the OpenFlow optical components in the high level design and a number of problems concerning OpenFlow integration are discussed, namely OpenFlow agent architecture, protocols to be used and cooperation with GMPLS software. It is generally assumed that OpenFlow should cooperate with ADVA's GMPLS control plane and this assumption is driven by the intention of control plane software development within OFELIA project. However it should be mentioned that other approaches of integrating OpenFlow into ADVA's NEs are conceivable. In [18] an idea to use OpenFlow to control TDM/WDM networks as an alternate technology to GMPLS was proposed. It assumes that for TDM/WDM networks flow can be identified by specifying: physical port number, wavelength or a time slot. Hence for WDM equipment flow table determines cross-connections setup (i.e. software and hardware).

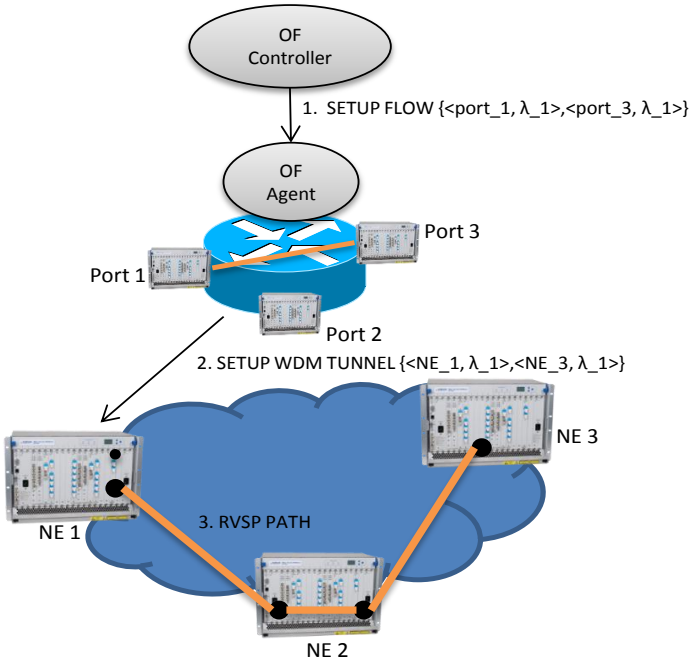


Figure 22: Cross-connect setup in a virtual switch

In discussions within Control Plane Team and with University of Essex a very general proposal of OpenFlow and GMPLS integration has been introduced. This proposal assumes that a network containing a number of NEs can be virtualized as an optical switch controlled by an OpenFlow agent as shown in Figure 22.
ADVA's GMPLS control plane utilizes OSPF-TE to distribute TE information and RSVP-TE to establish connections in data plane. GMPLS software has its own abstract representation of available resource and hides the complexity of managing those resources. Currently there is no open, standardized API for GMPLS that would expose resources in the OpenFlow protocol manner.

Given the two different approaches that OpenFlow and GMPLS represent, a problem of integrating them arises. As it was mentioned in the introduction, an idea to represent GMPLS-enabled network as a virtual optical switch has been proposed. In that concept, it is assumed that a NE can be treated as a port of a virtual switch which is managed through the OpenFlow protocol. OpenFlow controller communicates with an agent which can speak to NEs. OpenFlow controller can issue requests to create flows which are identified by specifying input and output ports and a wavelength. Flow definition constitutes a cross-connection within virtual switch. Cross-connection setup request can then serve as an input for GMPLS to setup a particular tunnel between NEs. GMPLS operation can be triggered by SNMP, CLI or other interface available on NE.
There is also a strong reason for virtualizing network as an optical switch and it comes from the fact that optical transport networks have fundamentally different nature than networks that OpenFlow was originally envisioned for. Network virtualization hides physical layer complexity and therefore allows the OpenFlow to remain simple and uniform control mechanism.

3.3.4.6.1 Virtual switch boundaries

In the OFELIA project, ROADMs are going to be used with external transponders. It means that an external transponder can be connected to either a non-reconfigurable filter or a reconfigurable one. Therefore virtual switch boundaries have to be placed on demultiplexed ports (i.e. supporting only single lambda), see Figure 24 and Figure 25 for expected cases.
Given a request to setup a flow that specifies a port number and a wavelength, like it was described earlier, a particular filter's port must be deduced from that request. It is necessary, because in order to setup a working tunnel every single device on a selected path must be properly configured and that includes filters. For filters that feature static mapping between wavelengths and ports it can be easily done. However in case of a reconfigurable filter (CCM) it cannot be deduced through which port a tunnel should be established. Therefore it seems to be necessary to change the notion of port addressing to encompass information about a particular filter port. This addressing may have format: x.y.z, where x identifies NE, while y and z refer to a particular filter device and a port number respectively. Each port may support only a single wavelength at a time, in case of a non-reconfigurable filter this is a specific wavelength (bounded to a given port) and one from a set of wavelengths in case of a reconfigurable filter. This concept also covers the case when virtual switch boundary is placed on a multiplexed port, in that case port may support a set of wavelengths at the same time.

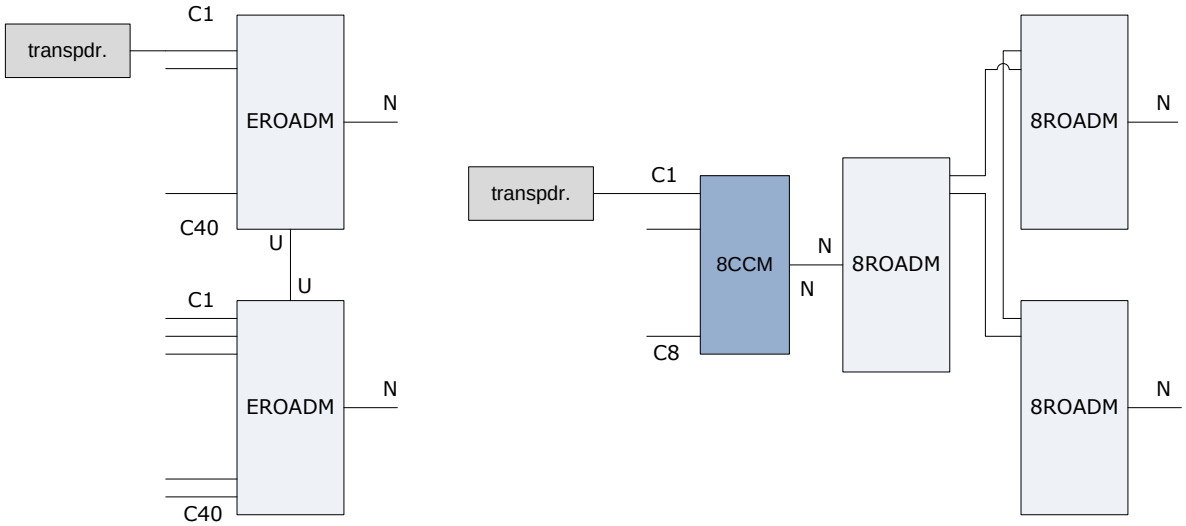


Figure 23: External transponder connected to an EROADM and to a CCM module

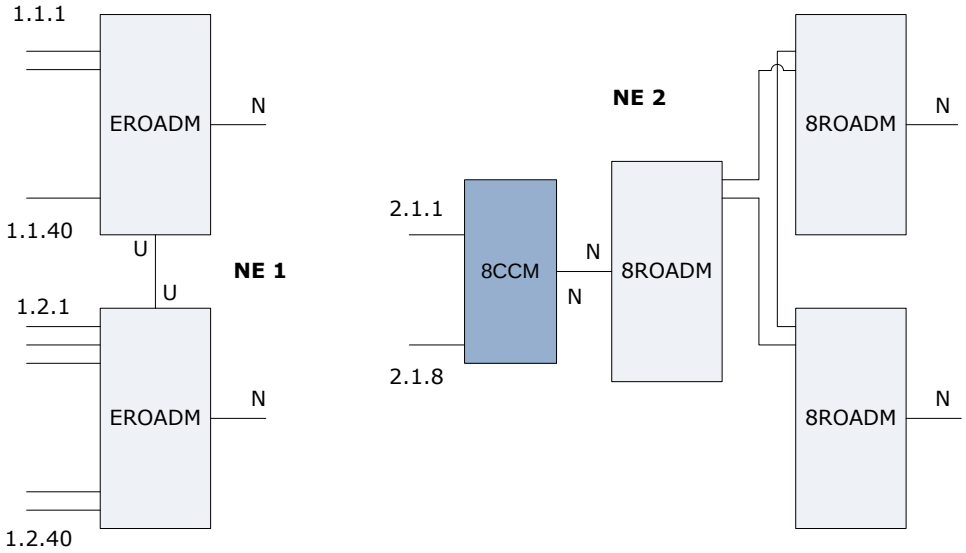


Figure 24: Extended port addressing – example

3.3.4.6.2 DCN configuration

It should be noticed that NEs must be connected through an IP network to utilize GMPLS as explained in the section 3.3.4.5. If external transponders are used and no OSC modules are available on NEs then IP connectivity can be provided be copper Ethernet. That probably means that NEs must be located within the same laboratory or building. NEs must also be configured properly prior to any further operations (e.g. IP addressing, TE links).

3.3.4.6.3 OpenFlow agent

OpenFlow agent is an entity that communicates with an OpenFlow controller on one side and with NEs on the other side. OpenFlow agent can utilize a number of interfaces to communicate with NEs:

- SNMP,
- TCLI,
- CORBA.

-

Choice of an actual interface (or interfaces) depends on the requirements, capabilities of an interface, agent architecture and location (e.g. application external to GMPLS servers or embedded) and it is also a matter of policy regarding exposing interfaces to third-parties. It should be also taken into consideration that available interfaces may not have required functionalities and therefore may need to be extended.
OpenFlow agent can be located externally to NEs, for example it may run on a separate server that has IP connectivity to NEs. It is also be conceivable that agent runs on the same box as NE software or even it's a part of the NE software. However in the OFELIA project it is agreed that an overlay model will be deployed, that is an agent will be located externally.
Among available interfaces, SNMP is the one that is open for third-party applications and gives remote access to the NE. Through SNMP, NE can be configured, modules can be provisioned and configured and also GMPLS can be controlled. TCLI and CORBA are proprietary interfaces and their usage would be considered only if the agent was an embedded entity developed entirely by ADVA.

3.3.4.6.4 Virtual switch model

OpenFlow agent must provide specific information concerning switch capabilities, e.g. number of ports and their parameters. In the OpenFlow protocol such information is exchanged between the agent and the controller by means of Feature Request/Response messages.
As it was described in section 3.5.6.1, when external transponders are in use, the notion of a port in a virtual switch needs to be changed. How should OpenFlow agent present information about available ports? One approach could be that each NE is represented as a single port in the OpenFlow but some additional information is required to inform controller about wavelength demultiplexing modules and their ports, such information could be added to a structure describing a port. In the other approach each filter's port can be represented as a port in the OpenFlow protocol. In the former approach the number of OpenFlow ports is equal to the number of NEs while in the latter it is equal to the number of filter's ports (i.e. it is much larger). However in the latter, it would be perhaps easier to exchange information regarding port's operational and administrative states or port's capabilities (available wavelengths). This problem certainly requires some further investigation.
It's conceivable that extended addressing artifacts (e.g. filter's or port's identification) are directly encoded in the OpenFlow messages and therefore OpenFlow controller should interpret them. However extended port addressing may also remain opaque to an OpenFlow controller. In that case addressing artifacts can be encoded into a single OpenFlow port field. The latter approach seems to fit better in the general OpenFlow concept. The actual filed encoding is a subject that requires further research.
Another virtualization problem is that whether agent should present to a controller information regarding all feasible cross-connects. Such information would tell OpenFlow application which lambdas from port X could be connected to which lambdas on port Y. This information could be extracted from TE database and it would reflect actual topology and result from switching capabilities of NEs. It's also conceivable that a map of feasible cross-connects can be provided statically, e.g. through a configuration file to the OpenFlow agent. The latter approach is much simpler to implement and it's easier to enforce policies regarding which paths can be exposed to the OpenFlow controller.
In the packet-switched OpenFlow there are practically no restrictions on creating flows, however it is a different case in transport networks, e. g. lack of wavelength conversion imposes a serious restriction. Therefore a map of feasible cross-connects would certainly be helpful to an OpenFlow applications. However the map of a feasible cross-connects can be considered as an optional feature and requests that don't satisfy physical hardware constraints can be simply rejected (e.g. path-computation engine would fail to compute a path).

3.3.4.6.5 GMPLS Co-operation

It is required that NEs are properly configured, so they can exchange GMPLS signaling and routing messages, moreover TE links must be configured prior to any OpenFlow related operations. OpenFlow agent requires IP connectivity to NEs but it also needs to know which modules (filters, ROADMs) should be treated as virtual switch boundary. It's assumed that the list of NEs IP addresses and a list of modules will be given in a configuration. Agent must also assign port numbers to NEs, modules and modules' ports, agent also has to maintain that mapping.

Given that OpenFlow agent has established a connection to a controller and they have exchanged information regarding virtual switch capabilities, an OpenFlow application may create, delete or modify flows. Flow modification request specifies a flow description (input/output port numbers and lambdas) and action (create, destroy). Three possible scenarios were identified on how the virtual switch operates the GMPLS objects:

- fixed (Stage 0) - the WDM tunnels are pre-established and OpenFlow agent manages their reservations based on Open Flow controller requests
- static (Stage 1) – OpenFlow agent manages pre-configured tunnels
- dynamic (Stage 2) – the WDM tunnels are established and torn down dynamically in response to appropriate OpenFlow requests

The first two scenarios, namely "fixed" and "static" assume much more simpler form of integration. In the first scenario it is assumed that OpenFlow agent can operate only on WDM tunnels that are already established on NEs. It means that agent can only monitor the state of an existing WDM tunnel and bind them with appropriate flow per request basis. Second scenario is very much like the first one, the difference is that OpenFlow agent can bring pre-configured tunnels up or tear them down. In both scenarios OpenFlow agent can easily collect information about configured tunnels and even create a map of feasible cross-connects. It means that modules can be created and configured prior to any OpenFlow operations, however for reconfigurable filters it still may be necessary to configure connected modules dynamically by OpenFlow agent to constrain path computation engine to use a particular wavelength.
In the third scenario when a request to create a cross-connection is issued, OpenFlow agent must resolve port numbers and retrieve NEs' addresses (ingress, egress) and modules' addresses (AIDs). Then it has to trigger path computation through appropriate management plane protocol in order to verify if virtual cross-connection in virtual switch topology is possible. In the next step GMPLS controller on the ingress NE is triggered again through appropriate management protocol to setup a tunnel and return the result. If the tunnel is established, agent must update the flow table. It should be noticed that flow table must contain tunnel AID. If a request to delete a cross-connection is issued then OpenFlow agent must resolve port numbers and NE addresses, check the flow table to retrieve tunnel AID and trigger tunnel deletion on the ingress NE. The steps in this scenario can be identified in a form of creation use case:

- OpenFlow client sends connection request to OpenFlow agent built by UEssex/ADVA,
- OpenFlow agent translates "ports" on virtualized "L1 switch" to network endpoints (e.g. node/module),
- OpenFlow agent triggers path computation to validate path between endpoints,
- OpenFlow agent triggers Tunnel establishment between endpoints,
- OpenFlow agent returns status to client,

and similarly deletion use case:

- OpenFlow client sends tear down request to OpenFlow agent,

- OpenFlow agent translates "ports" on virtualized "L1 switch" to network endpoints (e.g. node/module),
- OpenFlow agent triggers Tunnel tearing down process,
- OpenFlow agent returns a status to client.

In contrast to the packet-switched OpenFlow, in transport networks controller doesn't receive any packets through a "secure channel", hence there is no need to implement such feature in the agent. Flow modification cannot be packet-driven (refer to the learning switch tutorial for OpenFlow), that is OpenFlow application must explicitly request flow modification. It is an important conclusion in the context of packet-switched and WDM network cooperation.

Regardless of final architecture decisions fixed, static and dynamic scenario can be treated as Stage 0, Stage 1 and Stage 2 of the implementation of OpenFlow agent for Layer 1 / Layer 0.

3.3.4.6.6 Provisioning of transponders

An important issue is how the tunnels' establishment by GMPLS is going to proceed when external transponders are in use. Currently GMPLS software supports establishing tunnels from and to transponders provisioned in NEs. Therefore if a request from OpenFlow controller is issued to create a flow, particular transponders have to be available in ingress and egress NEs. Transponders can be provisioned:

- manually,
- automatically by the OpenFlow agent at startup,
- dynamically for a given request.

It should be noticed that OpenFlow agent needs also to configure transponders if necessary, e.g. provision a plug tuned for a particular wavelength. In fact provisioning of a plug may be useful to constrain path computation engine to use a specified wavelength.

3.3.4.7 Design features and considerations

In the integration design explained in section 3.3.4.5, it is assumed that the extended OpenFlow controller is participating both in the control plane related communication of the optical circuit switching domain (e.g., GMPLS) and also in the OpenFlow communications. However, the former is not a hard requirement. This means that the extended OpenFlow controller can implement its own procedures to participate in an interior routing protocol (e.g., OSPF-TE) and arrange the lightpath establishment and teardown procedures with all the intermediate optical connection controllers along the lightpath. Therefore the intermediate nodes should run the same protocol for exchanging routing information and also run the proper agent for establishing and tearing down the lightpaths.

Other important assumption in this design is the presence of OpenFlow enabled switch (agent) as discussed in section 3.5.3, which is required to get involved in the OpenFlow related communication with the extended OpenFlow controller for each new flow that is entered the optical circuit switching domain through the add/drop ports of the ROADMs. This OpenFlow enabled switch triggers the extended OpenFlow controller to establish the required lightpath in the optical circuit switching domain, before updating the flow-table entries of the OpenFlow enabled switches. This setup can be integrated in the ROADM's control logic. The integrated solution removes this need by assigning an OpenFlow enabled switch for each ROADM network element (and its corresponding add/drop ports).

As mentioned the proposed high-level design is the outcome of the initial discussions and collaboration with ADVA in order to enable the ADVA equipments with OpenFlow protocol. More detailed aspects of this approach and detailed design of the proposed integration scheme along with possible updates/changes will be reported in the next related deliverables.

3.3.5 Resource listing plug-in subtask

The resource listing plug-in is filling a gap of the current Expedient implementation. Experimenters cannot get an overview of available resources e.g., slices in advance. They can only select the resources when they create an experiment.
This plug-in will provide a resource list for browsing all available resources before creating an experiment and to check for the availability of an adequate quantity of resources.
The resource listing plug-in is still under development because the API's to the aggregate managers that provide the resources are not fully ready for use.

3.3.6 Opt-in manager improvements and bug fixing subtask

Opt-in Manager Aggregate Manager or simply "Opt-in" consists of a database that stores information about which users and experiments have been assigned to which part of the flowspace. When a user wants to join an experiment, it informs the FlowVisor and the user flowspace is merged into the flowspace of the experiment. A graphical user interface allows the reservation of certain flowspace and allows users to opt in to experiments.
For the base control framework, we rely on Opt-in manager from the GIT repository. Although, in principle this version fits our needs, we studied the implementation on three important aspects: check of intersection between the requested flowspace and the slice flowspace, flow auto-approval and detection of topology changes.

3.3.6.1 Intersection between slice flowspace and requested flowspace

The Opt-in manager receives flowspace request from the upper entities of the framework (i.e. Expedient). The goal here is twofold:

1) Make Opt-in manager able to check if these flow requests actually remain within the flowspace belonging to the slice (i.e., the flowspace which the slice has the right to control) and eventually grant a flowspace within the flowspace belonging to the slice and according to some policies.

2) Whenever there is a change in the flowspace belonging to the slice, recheck the differences between the new flowspace belonging to the slice and the one previously granted. Shrink or expand the flowspace to which the slice has access in the facility according to these changes in the flowspace belonging to the slice.

For achieving this goal two potential solutions are proposed:

1) Create a new Aggregate Manager, which would be in some sense higher in hierarchy so that it can inform the Aggregate Manager (the one currently used) and the Opt-in manager about changes in the flowspace belonging to the slice.

2) Communicate the Opt-in manager with the current existing Aggregate Manager so that the latest can inform the Opt-in Manager about the changes in the flowspace belonging to a particular slice.

On the side of the Opt-in manager, a new script of approval of flows would be needed. This script should be raised whenever a slice wants to be granted access to a flowspace in the facility managed by the Opt-in manager, but also whenever the flowspace belonging to the slice changes. It should compare the flowspace which is currently belonging to the slice with the flowspace requested to use in the facility (or with the flowspace already granted to use in the facility) and take the relevant actions.

In addition, in terms of implementation, a new API should be defined if eventually decided to communicate with the Aggregate Manager. Otherwise, SFA should be used to communicate with the higher hierarchical Aggregate Manager.
In order to accomplish this task it is a requirement to agree on a common slicing mechanism.

3.3.6.2 **Flow auto-approve methods**

For the sake of simplicity let us obvious the checking of the matching between the intersection of the requested flowspace and the flowspace actually belonging to the slice. It will let us concentrate on the issue of auto-approval of flows.
By design a root administrator initially owns the complete flowspace. For our testbed this means that an island manager would need to manually approve any request for a piece of flowspace by a user. In our OFELIA framework we already rely on Expedient as a platform to create and run experiments. Thus, it is desirable that the Opt-In manager does not always have to check again for potential collisions in the used flowspaces (if a certain piece of flowspace has already been allocated to a user or experiment by Expedient).
Let us give an example and assume that slicing, i.e., separation between different experiments is done based on MAC addresses. This is purely an example, as how slicing is going to be done is still a topic of discussion inside the consortium. Moreover, the Opt-in manager is aware of all MAC addresses that belong to a specific slice or experiment. If the Opt-in manager receives a request for all HTTP (port 80) TCP traffic, it should leave this request in stand-by-state and wait until the responsible island manager manually accepts or rejects this request. However, if the request includes all HTTP (port 80) TCP flows where the MAC source and destination addresses both fall into the pool of MAC addresses of the current slice, then it should be auto-approved by the Opt-in manager. If it requests more MAC addresses than the ones that belong to the slice, the Opt-in manager can decide to auto-approve only the flowspace matching with the intersection of the MAC addresses requested and the MAC addresses belonging to the slice.
Our task here was to check the availability and to test the functionality of the auto-approve methods.
To this end, we study the source code of the Opt-in manager and find in the folder /optin_manager/src/python/openflow/optin_manager/auto_approval_scripts a set of example scripts to define auto-approval of flows according to different rules. For example, the script approve_all.py allows to automatically approving requests for arbitrary pieces of flowspace. Other scripts allow to reject all requests automatically or to postpone all decisions for later manual approval. Moreover, we point out that the graphical user interface of Expedient provides an option to enable or disable auto-approval, or choose a customized policy to approve or reject flowspace requests, and relies on the just mentioned scripts.
Testing some of the provided scripts, we confirm that the auto-approve methods work without any problems, according to the policies specified in each script. Based on the provided code examples, it does not appear difficult to define our own auto-approval policies for the OFELIA testbed, or even different policies for different islands that could eventually be implemented in a per-island fashion.

3.3.6.3 **Learning about topology changes**

During our testing of the base control framework, we have realized that topology changes (e.g., additional or failing connections, additional devices) do not become visible in Expedient. Our task was to find potential explanations for this problem and to scrutinize the general mechanisms for propagating topology changes towards Expedient. Overall, there seems to be two different mechanisms how Expedient can learn about changes in the topology, illustrated in figure

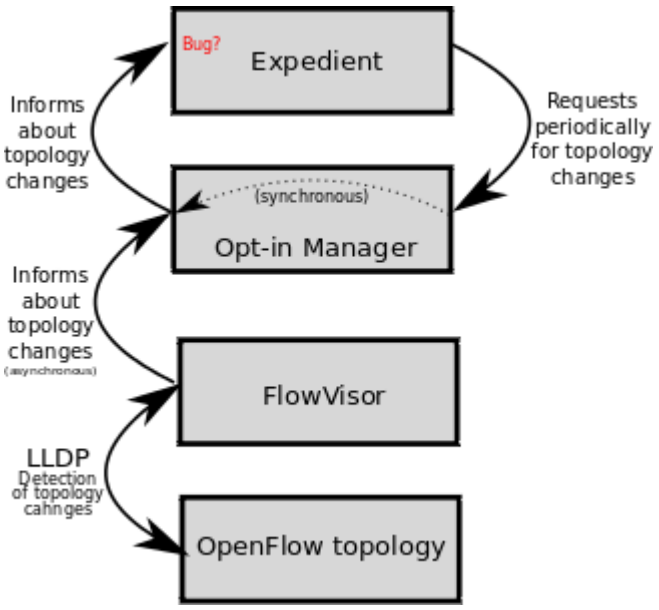


Figure 25: Propagation of topology changes

First, the FlowVisor component runs the Link Layer Discovery Protocol (LLDP), a link layer protocol that allows network devices to announce their existence and capabilities. In contrast to Expedient, the FlowVisor component generally resides inside the OpenFlow network and it is therefore the ideal candidate for detecting potential topology changes. FlowVisor informs the Opt-in manager in the case of topological changes. The Opt-in manager just forwards this information to expedient, through the callback implemented in the script fv_api.py within the directory optin_manager/src/python/openflow/optin_manager/xmlrpc_server/ of the source code of the Opt-in manager. After performing some tests, we believe that the mechanism does not work correctly due to bugs within the Expedient software, which prevent Expedient from showing the updated information.
Second, as pointed out by the developing team of Stanford University, Expedient itself can periodically try to find about potential topology changes. To this end, it asks the Opt-in manager at fixed time intervals to provide information about the current topology. However, this feature is only available in the latest version of Expedient (version 4.0.0) and could not be working properly, probably because of the same bug which encumbers asynchronous updates.

3.3.7 Integration tests, debugging and Expedient's GUI improvements subtask

The Software taken from Stanford served well as a basis for OFELIA's Phase I. It offered much functionality right of the shelf and made it possible to adapt rather than re-implement. As Phase I strived to deliver a running system, Expedient and the Opt-in Manager were good software packages to start off with.

As part of the OFELIA's methodology, it became clear, that the objective of Phase I was to gather as much experience as possible from existing architectures. The design decisions and structure, as well as the complexity of the software could be assessed during the adaption of Stanford's software. The learned assessments and lessons will serve as a basis for the architecture of Phase II:

Coupling and cohesion

The code of the two software packages, Expedient and Opt-in Manager, are bundled together very strongly. One example is the usage of the same database. This tight coupling made it very hard to firstly understand the code, secondly to debug the code and thirdly to adapt the code. Additionally, the cohesion (semantic measure of how related the functionality of a module is) of the classes and modules is very low. Many classes fulfill different semantic responsibilities; hence make it hard to

make sense of the code. Since the above characteristics are not favorable, the structure of OFELIA's Phase 2 software will focus on decoupling and semantic separation.

Code structure

Even though the two software packages are implemented using the same framework Django, the structure of the code varies among them and even within the modules. Since Django does not confine the programmer in structural matters, it became clear that conventions need to be established for Phase 2. With those conventions, regarding file structure, classes to be used and coding style, the project will become easier to understand and maintain. Before Phase 2 starts, a Python and Django tutorial should be held to establish a common understanding of the conventions among all development partners.

Complexity

As discussed above, the software packages are difficult to comprehend and maintain. This is not only due to the lack of design principles. The main reason is that the implementation of the two software packages are quite complex in terms of functionality and technologies used. The major lessons to be drawn from these findings are: the design of Phase II's architecture needs to as simple as possible and the number of technologies and libraries used needs to stay at a minimum.

Usability aspects

In Phase 1 the design elements (e.g. logo, colors) of Expedient and the Opt-in Manager were adjusted to OFELIA's project identity. Yet, the current interface is aligned to the needs of Stanford's requirements. The workflow of Expedient works fine for OFELIA's purposes, but is not as slick as it should be. One example is the addition of Aggregate Managers (AMs) to a slice. In Expedient it was necessary to first add an AM to the Expedient, then add it to a project and then to a slice. This lengthy workflow is not necessary for OFELIA, so the step of adding it to a slice was automated, hence simplified. Another lesson to be learned from Phase 1 is the user-unfriendly interface concept to have multiple instances of the same form on one page. This concept is not very common in the WWW and proved to be very difficult for users. Hence, the user interface of Phase II needs to be intuitive and as close to common usage patterns as possible.

Since Phase 1 was focusing on adjusting the functionality of the code, the workflow of the system was not changed significantly. Rather, the interface design for Phase II will be done from scratch before adjusting/porting the existing code. Having the lessons learned from Phase I in mind, screen sketches will be drawn and used as a basis for implementation.

4 Current state of the implementation

This section is meant to give an overview about the current substaks that are being developed under the task 5.2 as well as the current state of the code. It also contains the future plans for each of the subtasks and the goals that need to be achieved.

4.1 LDAP integration subtask status

The current implementation of the integration with LDAP has been done and is placed in the branch LDAP. However it has not been merged into the code release at the time these lines are being written, since the control framework LDAP has been deployed but not yet configured.

Therefore, the code needs deeper testing against production LDAP and slight modifications in order to meet the requirements.

4.2 Server Virtualization software subtask status

From the objectives mentioned in section 3, the current status of some tasks is detailed below. The unmentioned here are implemented if they were short term objectives or not yet implemented if they were for long term.

- Regarding the virtualization technology, Xen is the only one supported.
- There is only a unique VM file-disk image template with a clean Debian Lenny distribution. The template image with all the Openflow tools is not yet build. VM upload from users is not implemented.
- User administration and authentication is done in each module (Expedient, aggregates) locally using a correspondent MySQL database. First tests to add LDAP support in Expedient have been done with positive results, but not fully complete since there was no LDAP server to test against.
- Deleting a VM machine in the server is not yet implemented in the Agent.
- The communication has been tested and is functional.
- From the Expedient it is possible to create, start and stop (also delete when function is implemented in the Agent) VMs in the server.
- From the VT AM, the IM is able to create, start and stop (also delete when function is implemented in the Agent) in the server, but not able yet to communicate back to the Plug-in.
- IP and MAC allocators in the VT AM are functional
- The traffic isolation and the IP/MAC spoofing prevention inside the server is not yet ensured. Tests are being conducted using an OpenVSwitch instance to connect Xen's dom0 eth0 interface with the virtual interfaces of the domU VMs in order to map VM MAC address with switch port.

4.3 Adding support for ProtoGENI-enabled equipment subtask

A 2nd Virtual Wall at IBBT is currently under test. Software on this Virtual Wall has been very recently migrated to the latest Emulab version and enabling the ProtoGENI AM API included in that latest EmuLab version is still ongoing and should become fully operational very soon. Once achieved, we will be able to test the full AM API of the ProtoGENI interface. Once the test phase of this Virtual Wall is completed satisfactory, the 1st Virtual Wall (to be used by OFELIA) will also be upgraded to this EmuLab version.

Therefore until now we have tried to connect the Expedient with a reference implementation of the ProtoGENI Component Manager (CM) API on a separated dedicated setup (in a virtual machine). There are two key problems with this setup:

1. Although the reference CM implementation is made available as a package on its own, it actually requires to be embedded in a full Emulab setup. Thus testing with / developing against a standalone CM implementation is far from ideal.
2. This reference CM implementation is not Component instead of an Aggregate Manager API. Normally the Expedient plug-in to be developed needs to be communicating with the AM and as a result of that, the AM communicates with the CM in order to delegate the requests.

So far, we have been able to have the PlanetLab plug-in communicate to a very limited extent with the ProtoGENI CM reference implementation. By slightly modifying the PlanetLab plug-in we have indeed been able to retrieve the response from the GetVersion function API call: the fact that these modifications were needed is probably a result from the fact that we were calling the CM rather than the AM API. Unfortunately, any of the other function calls (as the DiscoverResources) from the ProtoGENI CM API resulted in errors.

On the other side, we have set up the Reference GENI Control Framework (GCF) which provides a dummy sample of a AM and clearinghouse. It turned out that out of the box, connecting the PlanetLab plug-in in Expedient to register an aggregate worked pretty well. However, as this Reference GCF is a dummy instead of a PlanetLab or PortoGENI implementation and thus further functionality could not be tested.

Despite the effort spent in trying to get the Expedient to connect to the ProtoGENI CM API Reference implementation, the latter experiment does not suggest we should be pessimistic about connecting the PlanetLab plug-in in Expedient to a full functional ProtoGENI AM (instead of CM) API and thus about the feasibility to develop a ProtoGENI specific modules starting from that PlanetLab plugin.

Future Work:

According to the above discussion, we will try to get the PlanetLab plug-in in Expedient to properly connect to an operational ProtoGENI AM API and assuming this will be as easy and as successful as the test with the dummy reference GCF implementation, start modifying the PlanetLab plug-in code to process the ProtoGENI RSpecs in order to turn it into a real ProtoGENI plug-in.
In case we do not succeed to realize the basic SFA based communication between the PlanetLab plug-in and the ProtoGENI AM API, we still have a fallback solution to implement a plug-in targeting the Emulab proprietary XML-RPC API. This API would require the Expedient plug-in generating a ns-script file similarly as the Emulab Web User Interface generates or can accept for uploading. Experience with the proprietary XML-RPC API is available, implying a lower risk.

4.4 ADVA's optical equipment adaptation to Openflow subtask

Based on initial discussions between ADVA and University of Essex an initial high-level design for adapting OpenFlow to ADVA ROADM has been done. Hardware configuration of ADVA ROADM was decided and built around the FSP3000 ROADM. It is focused on the proposal of virtualizing network as an optical switch managed by one OpenFlow agent. SNMP was highlighted as an interface through which agent should communicate with NEs. SNMP seems to be a reasonable choice, as it has required functionality and it is opened for third-parties. Proposals for interlayer operation and slicing was also done. As next steps, we consider two approaches to integrate OpenFlow capability into ADVA optical switches. The first one is an Overlay Model where the OpenFlow agent will reside outside the ADVA ROADM and the second approach is an Integrated Model in which the OpenFlow agent will be built into the box. It worth mentioning that these integration schemes will become more detailed (and possibly updated) as we move forward in the detailed design and development phases of the project.
For the virtual switch proposal, which is the overlay model of OpenFlow integration, three stages of deployment were identified and described in 3.3.4.6.5, namely:

- stage 0 – fixed scenario,
- stage 1 – static scenario,
- stage 2 – dynamic scenario

Each stage is meant to be more complex and enclose more functionality than the previous ones. In the next step details concerning:

- particular use cases (communication flow),
- configuration (data required for the agent to operate),
- and protocols (SNMP and OpenFlow for L0/L1 layers, encoding),

will be formulated.

4.5 Resource listing plug-in subtask

The resource listing plug-in is still under development because the API's to the aggregate managers that provide the resources are not fully ready.

4.6 Opt-in Manager improvements and bug fixing subtask

In terms of the two studied aspects (flow auto-approve methods, learning about topology changes) we do not expect any difficulties for use within OFELIA. With respect to auto-approve methods, the available code examples provide useful hints to implement our own policies. The fact that topology changes do not become visible within Expedient should not be an issue if we switch to the latest version of Expedient. For the further development of the Opt-in Manager it will be crucial to reach a decision on the used slicing mechanism (based on MAC addresses, VLANS, IP addresses, etc.), and narrow down the requirements.

4.7 Integration tests, debugging and Expedient's GUI improvement subtask

Plans for Phase 2

The lessons learned and principles identified in Phase 1 will serve as foundation for decisions in Phase 2. As the current implementation lacks appropriate structure, Phase 2's implementation will start from scratch and add previous code iteratively. The steps will be:

1. Identify the major functional requirements(rough requirements analysis) and prioritize them(e.g. "handle slicing", "start VMs", "authenticate users", "monitor experiments", "be compliant to SFA")

2. Create a rough concept of the overall architecture and necessary interaction patterns
3. Hold tutorial to introduce structure and conventions
4. Refine the concept for the functionality with the highest priority
5. Draw sketches of the UI (and get feedback regarding usability)
6. Implement the refined concept; Reuse previous code where applicable
7. Refractor code where necessary
8. Iterate from step 4

Due to the complexity of the system, the method above is modeled after Agile approaches. The process of finding all requirements and establishing all specifications was attempted in Phase 1. It was found, that the functional modules need to be regarded separately and refined iteratively.

5 Description of the software deliverable

The current deliverable includes a digital part of the deliverable that can be downloaded in the following URL (the exact details, like access password, will be sent apart):

`http://www.fp7-ofelia.eu/storage/`

For the purpose of an easy and simple evaluation of the development efforts that have been taken place in T5.2, the latest release of code has been configured and installed in a modified version of the SDK. These two machines contained in the file, are nearly self contained, and can perform all the actions with the exception of VM provisioning actions, that require a XEN server[3]

The file contains two virtual machines, as the usual SDK:

- Mininet VM: Containing mininet software to emulate openflow-enabled switch topologies, as well as a NOX controller installed.
- Control framework VM: This VM contains an installation of Expedient, Opt-in Manager AM and Virtualization AM.

Steps to be installed and tested:

1. Install Virtualbox
2. Import both appliances (under File>Import Virtual appliance)
3. Boot them (username is openflow, password openflow).

The following table contains the information required to access to the different component's GUI:

Component	VM	URL	Administrator level username/password	User level username/password
Expedient	Control Framework	**https://127.0.0.1/**	expedient/expedient	user/openflow
Optin Manager	Control Framework	**https://127.0.0.1:8443/**	expedient/expedient	-
Virtualization Manager	Control Framework	**https://127.0.0.1:8445/**	expedient/expedient	-

In order to perform Openflow tests, the mininet VM must be up and running with connectivity between both machines (only L2 connectivity is required, IP configuration is already done).

[3] If required, please issue an email to marc.sune@i2cat.net, in order to get the details of an available XEN server to perform tests against it.

6 References

[1] OpenFlow 1.0.0 Reference Implementation, http://openflowswitch.org/downloads/openflow-1.0.0.tar.gz

[2] Expedient web page, http://yuba.stanford.edu/~jnaous/expedient/

[3] Opt-in Manager, http://www.openflow.org/foswiki/bin/view/OpenFlow/Deployment/HOWTO/ProductionSetup/UsingOptInManager

[4] Xen Homepage, http://www.xen.org

[5] SFA, http://groups.geni.net/geni/attachment/wiki/SliceFedArch/SFA2.0.pdf?format=raw

[6] GIT Homepage, http://git-scm.com/documentation

[7] Protogeni web page. http://www.protogeni.net/trac/protogeni

[8] OpenNebula , http://www.opennebula.org

[9] Mininet Walkthrough, http://www.openflow.org/foswiki/bin/view/OpenFlow/MininetWalkthrough

[10] NOX Manual, http://noxrepo.org/manual/using.html

[11] Django Web Framework, http://www.djangoproject.com/

[12] xmlrpclib, http://docs.python.org/library/xmlrpclib.html

[13] rpc4django, http://davidfischer.name/rpc4django/

[14] GenerateDS, http://www.rexx.com/~dkuhlman/generateDS.html

[15] VirtualBox, http://www.virtualbox.org/

[16] OpenVSwitch, http://openvswitch.org/

[17] S. Das, G. Parulkar, N. McKeown, "Simple Unified Control for Packet and Circuit Networks," IEEE Photonics Society Summer Toplical on Future Global Networks, July 2009.

[18] S. Das, G. Parulkar, N. McKeown, "Unifying Packet and Circuit Switched Networks with OpenFlow".

[19] R. Sherwood, G. Gibb, K-K. Yap, G. Appenzeller, M. Cassado, N. McKeown, G. Parulkar, *FlowVisor: A Network Virtualization Layer.*

[20] OFELIA Milestone MS21 "Report on initial requirement study and use case analysis", R. Jayakumar , R. Nejabati, S. Azodolmolky, et.al. December 2010

[21] OFELIA Milestone MS51 "Software development environment setup", M.Suñé et.al., November 2010

[22] 8th GENI Engineering Conference (GEC), online: http://www.geni.net/?p=1739

[23] 9th GENI Engineering Conference (GEC), online: http://www.geni.net/?p=1878

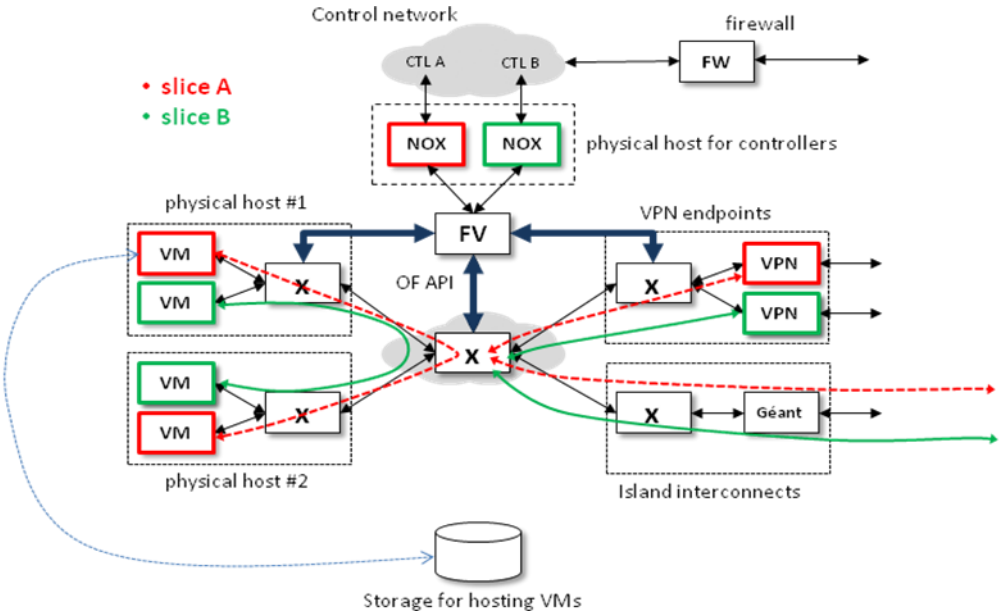
[24] Openssh-lpk project, online: http://code.google.com/p/openssh-lpk/

[25] Linux-Pluggable-Authentication-Modules, online: http://www.kernel.org/pub/linux/libs/pam/

Appendix A: Basic use-case document

Use Case	
Scenario/Use Case name	Basic functional tests for the OFELIA control framework
Organisation Involved	All OFELIA partners involved in implementing the basic OFELIA control framework.
Summary Scenario (storyline)	Overview: This use case defines the basic functional tests to verify proper operation of the OFELIA control framework, its OAM mechanisms, and the interface towards the user. • Two independent researchers A and B create experimental slices within OFELIA for conducting OpenFlow based experiments. • Each slice is isolated from others using a logical MAC based isolation scheme. • The user deploys at least two virtual machines as data sink and source within his slice and located inside a single island (phase 1) • The user deploys at least two virtual machines as data sink and source within his slice located in two different islands (phase 2) • The user deploys a NOX instance on a virtual machine created by the user inside OFELIA facility (like the case of an endpoint) (phase 1). • The user deploys a NOX instance on a machine outside of the OFELIA facility (phase 2). • The user deploys a logical VPN endpoint and establishes a connection between this VPN endpoint and a test bed outside of the OFELIA facility (phase 2). • The OFELIA facility must ensure proper isolation among slices by implementing adequate monitoring functionality (phase 2/3). • The OFELIA facility detects misbehaving slices and takes appropriate countermeasures (throttling, user notification, slice termination etc.) (phase 3). • Both users agree to unify their slices. OFELIA provides an online reconfiguration to attach both slices to each other (e.g. merging of functionality, facility reflector for exchanging traffic, etc.) (phase 3). • The user disables his slice and saves the experimental environment for later reuse (phase 2). • The user saves a virtual machine on an external storage (phase 2). • The user is allowed to deploy self-created VMs on a physical machine capable of hosting the VM inside the OFELIA facility (phase 3). Please note, that we expect this user case to evolve along the further development of OFELIA's control framework. It should be adapted to the requirements defined for each phase of OFELIA as a working use case.
Goal	The OFELIA control framework will evolve in three phases during the project's lifetime. The set of requirements of this control framework will be extended in each phase. This use case defines a scenario that allows verification of the basic set of requirements for each phase and defines the minimal functionality that must be provided by OFELIA. Focus in this use case lays on the proper operation of the control framework 1) to define logical slices for experiment isolation, 2) to deploy and manage virtual machines serving as host for OpenFlow controllers and data sinks and sources, and 3) to deploy and manage VPN endpoints allowing users to access their slice and/or to connect their slice to a network outside the OFELIA facility. The OFELIA control framework is accompanied by operation and maintenance functionality and a sophisticated interface towards the user. While the latter enables the user to define his testing environment and to conduct experimental control, OAM ensures detection of misbehaving slices to ensure proper isolation and to mutually shield slices from each other.

	The following list comprises all functionality that must be available and fully operational in the OFELIA facility. **Operation and Maintenance** **1. Topology detection (phase 1+2)** 1.1. Topology detection including a proper description and database (some form of a network description language) (phase 1) 1.2. Automatic updating of island topologies and overall facility topology when changes to the facility occur (addition or removal of new hardware) (phase 2) **2. Monitoring subsystem (phase 2)** 2.1. Detect hardware failures (disconnected physical links, device outages, etc.) 2.2. Detect CPU load on physical machines caused by deployed VMs. 2.3. Monitor amount of traffic generated by each slice to detect overloading and congestion situations 2.4. Monitor traffic generated by each VM directly on physical host 2.5. Monitor proper operation of control plane entities (UI, flow visor, switches, VPN controllers, VM controllers) **3. Notification subsystem (phase 2)** 3.1. Check notification subsystem to inform user about misbehaving a slice component (e.g. generating too much traffic) **4. Isolation enforcement (phase 2+3)** 4.1. Terminate misbehaving slices and notify user (phase 2). 4.2. Enable physical machines hosting VMs to throttle traffic emerging from a specific VM (phase 3). **5. Operations (phase 1+2)** 5.1. Verify proper connectivity of relevant control entities within an OFELIA island and within the overall OFELIA facility (phase 1) 5.2. Connect to notification subsystem (phase 2) 5.3. Monitor island interconnecting links (phase 3) **Facility user service** **6. Graphical user interface (phase 1+2)** 6.1. Topology depiction of islands/facility entities and interconnecting links (phase 1). 6.2. VM management (creation, removal, storage initiated by user) (phase 1). 6.3. Experimental control: read online status of slice for experimental control and make available to user in real-time ➔ slice topology (phase 2). 6.4. Dynamic reconfiguration of a testing environment by user (adding VMs, VPN endpoints, etc.) (phase 2). **7. XML-RPC application/script for controlling slice setup and operation (phase 2)** 7.1. Create an application/script to be able to send querys to the XML-RPC interface published by Expedient, for non-interactive control of the slice. console based control of user slice including setup and experimental control thus extending the user interface (phase 1). **Control plane functionality** **8. Basic slicing functionality (phase 1)** 8.1. Verify proper MAC address management per slice (how to select slices and how to make sure that no duplicates exist) 8.2. FlowVisor management and configuration (basic CRUD functionality for slices) 8.3. Configuration of slices must survive restart of FlowVisor **9. VM management (phase 1)** 9.1. Basic CRUD functionality for VMs including VM management and access 9.2. Adding/removing a VM to/from a slice 9.3. Accessing a VM via secure shell 9.4. Management of VM user credentials and basic configuration (like IP-addresses)

User community	Facility users, NOC personal, island managers
Figure to visualise the Use case scenario	
Virtualization layer and involved network/IT resources	This use case follows the basic connectivity abstraction of the OFELIA facility, i.e. a L2 based environment per slice is assumed. Later extensions towards transport networks may require an update to this use case. This is for further study.
Hardware Requirements	None. This use case focuses on basic resources required for all OFELIA facilities.
Access method to the Infrastructure (OFELIA Islands)	Secure shell access to virtual machines per slice. VPN endpoints for connecting a slice to non-OFELIA controlled networks and test beds. Various VPN types may be adopted.
Access Method to the Infrastructure Slice	To be decided within the ongoing architectural discussion among WP2 and WP5. This is an architectural detail, partially out of scope of a use case.
Openflow Controller Requirements	None.
Monitoring Requirements	Proper monitoring for OAM required including load on physical hosts and on intra- and inter-island connections.
Life-cycle of Service (Phases from start to end)	This use case is designed to cover OFELIA's overall life time, covering at a minimum the project's duration (September 2010 to August 2013).
Preconditions	Not applicable.
Post conditions	Not applicable.
Other comments	None.

