Federation of Community Networking Testbeds

Gerard Marin Nogueras

Master of Science Thesis
Stockholm, Sweden 2014

TRITA-ICT-EX-2014:092

Master of Science Thesis

Federation of Community Networking Testbeds

Gerard Marin Nogueras

Supervised by
Prof. Leandro Navarro (UPC)
Prof. Johan Montelius (KTH

June 2014

Master of Science Thesis
Software Engineering of Distributed Systems Master's Programme
School of Information and Communication Technology
Royal Institute of Technology (KTH), Stockholm, Sweden

Developed at Universitat Politèncinca de Catalunya (UPC),
Barcelona, Spain.

Supervised by Leandro Navarro (UPC) and Johan Montelius (KTH).
Examiner: Johan Montelius

Abstract

Testbeds have become an important facility for the Future Internet Research and Experimentation since they are a good stage between simulation and production covering areas of the future Internet like Community Networking. The range of possibilities offered by testbeds can be extended using federation. Federation enables the interoperability among testbeds, thus building an aggregate of facilities in which capabilities are combined and complemented to support new and more complex research and experimentation. Besides many advantages however, federation also brings challenges. In the thesis such challenges are identified and addressed for the concrete scenario of Community Networking testbeds. In particular the thesis addresses the federation of Community-Lab, a testbed facility for experimentation with network technologies and services for community networks. The main contribution consists of the development of a software tool to enable the interoperability and federation of the Community-Lab testbed with other testbeds.

Acknowledgements

I would like to thank Leandro Navarro for all his help, support, feedback and advise throughout the development of the thesis. Working under his supervision has been a pleasure. My grateful to Johan Montelius for his supervision as well.

I would also like to thank all the people who have helped or contributed somehow to this work. In particular, it was an invaluable help to have the Confine-ORM tool, thanks Marc.

Finally, my gratitude and admiration to my friends and lab-mates for so many experiences together. Thanks Ester, Mennan, Manos, Vamis, Navaneeth, Amin, Davide, Óscar, Leila and Esunly.

Contents

List of Figures

List of Tables

Table of Acronyms and Abbreviations

FIRE	Future Internet Research and Experimentation
GENI	Global Environment for Network Innovation
CONFINE	Community Networks Testbed for the Future Internet
C-Lab	Community-Lab
VCT	Virtual CONFINE Testbed
CONFINE-ORM	CONFINE Object Resource Mapper
CN	Community Network
CD	Community Device
RD	Research Device
SFA	Slice-based Facility Architecture
AM	Aggregate Manager
CM	Component Manager
R	Registry
SM	Slice Manager
SA	Slice Authority
RSpec	Resource Specification XML file
URN	Uniform Resource Name
HRN	Human Readable Name
XRN	Refers to any of the both types, HRN or URN

1 Introduction

1.1 Problem

The problem that this thesis aims to solve is essentially an interoperability problem. Community-Lab, an experimental testbed for research deployed inside several community networks to be federated with other experimental testbeds so that they can interoperate with each other and allow usage with a common, generic and standard application programming interface (API). To carry out such federation the testbed must be compliant with a selected standard architecture and API, thus allowing the testbed to correctly interact with other standard-compliant testbeds.

1.2 Goal

The main goal of the thesis is to develop a software layer on the Community-Lab testbed that enables federation. As said, the software layer will expose a standard-compliant interface for the testbed so that it can be federated with other testbeds and used for experiments by testbed agnostic tools using a common standard API. The thesis includes the study of the different possibilities for the analysis, design and development of federation support for the testbed.

Other objectives of the thesis include an analysis and reflections about the concept of federation in a wider context. Possibilities, benefits and limitations of testbed federation are explored, as well as a discussion about other potential federation domains for the specific testbed that the thesis addresses.

1.3 Methodology

The methodology followed for carrying out the thesis and its tasks consists of different phases: analysis, design, implementation, and validation.

The first part is basically a study of state of the art in the area of testbed federation. Such analysis includes background information on the testbed architecture and federation related concepts and tools, in particular for the addressed testbed. A proposed standard for federation of Future Internet Research and Experimentation (FIRE) facilities, which was already and successfully used in other projects, is studied in detail to understand the feasibility of its adoption as a solution for the particular testbed.

After the analysis, the acquired knowledge is applied to make decisions about the design and architecture of the software federation layer. After design the implementation phase starts. During the implementation, concepts and tools for the particular testbed are applied into the software development process. The code is implemented using a granularity that enables the separation in rather disjoint functional modules so that they can be tested in parallel during the implementation process.

The last phase of the process is the evaluation of the implemented software, integrated with the Community-Lab testbed, to verify its correct behaviour.

Different tests will be carried out during the evaluation process.

1.4 Validation

Validation of the implemented software tool is understood as a key aspect in this thesis as this federation service becomes part of the Community-Lab testbed and provides the federation API. A good and complete process of validation is considered necessary to guarantee a correct behaviour and the fulfillment of the requirements. Therefore, the validation process consists of an evaluation including different tests such as functional test from the end-user point of view, automatic functional tests and a reference experiment test. The validation phase allows to ensure the correct, expected behaviour of the software as well as its quality. It also includes a discussion about maintainability and future extensions of the implemented software.

1.5 Outline of the thesis

The thesis is structured as follows:

Chapter 2 presents the background to the work, with several subsections that situate the context and introduce testbed and federation related concepts relevant for this thesis.

Chapter 3 describes the federation scenario for the particular testbed addressed in this thesis and includes a discussion about potential federation domains for such a testbed.

Chapter 4 presents the Design of the software. This is the first chapter including details about the selected solution model and how such model is applied on the specific testbed.

Chapter 5 contains a description of the Implementation process of the software component that enables the federation of the testbed. The implementation is presented with considerable amount of detail to fully understand the process and the decisions made throughout it.

Chapter 6 presents the Evaluation of the implemented software, describing in detail the different tests performed to validate the correctness of the solution.

The conclusions and a final discussion are presented in Chapter 7. This chapter reflects on the work done, the results obtained, and the lessons learned throughout its realization.

The Appendix contains a manual for the user of the software. The manual can be seen as a guide that provides the details to use and understand the implemented software, from the installation and deployment to the usage and its potential. Writing such end-user manual was considered as a necessary task to deliver a complete piece of engineering work.

2 Background

This chapter presents the necessary background for the thesis. The section 2.1 addresses the need for testbeds in the area of community networking and the section 2.2 addresses the concept of federation as it is understood in this thesis. The remaining sections of the chapter introduce respectively the CommunityLab testbed, the Fed4FIRE project, the SFA model and the SFAWrap software, which all serve as a context and tools for the development of the thesis.

2.1 Community Networks and Testbeds

Community Networks are an emerging model for the Future Internet. Community Networks are defined in [1] as *"large scale, self-organized and decentralized networks, built and operated by citizens for citizens".* Such networks appeared as an alternative low-cost connectivity solution for underdeveloped countries and isolated areas. However, these kind of networks have become an interesting research facility and they are nowadays developed in rural and urban, rich and poor areas across the world [1]. Examples of community networks in Europe are: Guifi.net (Catalonia) [2], AWMN (Athens) [3], FunkFeuer (Wien) [4].

Community networks are dynamic and diverse, as they are composed of several types of links and nodes and offer different contents and services. Participation to a community network is open to anyone, which leads to a self-growing in topology and capacity based not only on planned deployment but also on immediate, unplanned demand [5]. Thus, community networks show a highly heterogeneous, unique nature even with a certain level of unpredictability on their behaviour. The mentioned characteristics make these networks hard to emulate and makes research results difficult to validate and uncertain. At this end, experimental testbeds become a suitable facility to support the research and experimentation in the area of community networking.

A testbed can be seen as a platform for experimentation that provides infrastructure and resources for running experiments in a rigorous, transparent and repeatable way. In particular, the testbed can be developed on top of a community network so the mentioned emulation difficulties are overcome. Thus the testbed becomes a platform for experimentally driven research to support the growth of community networks and improve their scalability and sustainability [5].

2.2 The concept of federation

2.2.1 Definition

The concept of *federation* is widely used in the context of communication networks. It was already introduced in the 90s, referring to the capability of interconnecting different transport networks to exchange data according to a common agreement [6] and since then the concept has evolved together with the Internet and communication technologies.

Current Future Internet research trends have revealed the need for federation, for example to guarantee the scalability of distributed systems growing in a non-planning fashion and to build new complex services by composing existing services that span multiple domains [7] [8] [9]. Therefore, interoperability, collaboration, coordination and sharing among different domains are necessary. Such new trends have contributed to refine the original definition of the term. Although multiple definitions exist, depending on the context in which federation is applied, there are some common key points that can be identified [10].

In particular this thesis addresses the federation of testbeds. A suitable definition in such context is given by Panlab [11]:

"A model for the establishment of a large scale and diverse infrastructure for the communication technologies, services, and applications and can generally be seen as an interconnection of two or more independent administrative domains for the creation of a richer environment and for the increased multilateral benefits of the users of the individual domains"

Federation can be seen as a solution for interoperability of facilities while keeping their scalability and autonomy. In other words, it refers to the ability of multiple systems to interact and work together to achieve a common goal. In particular for testbeds, federation helps building a global pool of facilities that can be accessed and used transparently through a unified, standard interface, thus building a global, larger and more complete testbed.

From the definitions given common points about federation can be identified. Federation implies a cooperation among federated components, which usually refers to the ability of federated components to leverage capabilities offered by other components in the federation. Capabilities offered depend on the specific domain and include for example infrastructure, resources (particular case for testbeds), services or applications [10]. Such sharing of capabilities is transparent from the user's point of view. Transparency implies that a user does not need to be aware of which federated component is offering a particular capability; from the user's point of view the federated system behaves as a single system with a standard way of accessing any of its components and capabilities [6] [8].

Although federation enables interoperability among components, they keep their autonomy to operate in isolation. Federation does not imply losing autonomy in terms of self management and control for the federated components [7]. Therefore, components continue working correctly on their specific domains.

2.2.2 Federation and experimental infrastructures

Federation is becoming a very important concept in the context of Future Internet Research and Experimentation (FIRE) facilities. The range of FIRE facilities is wide and includes a complex, heterogeneous variety of facilities and tools, from Internet communities and networks to clouds, services and end-user applications. Although most of these facilities are designed to work in isolation, like testbeds, the true potential arises from their combination. By mixing and integrating the different resources and capabilities offered by the different facilities, a larger, diverse platform can be build, allowing to bring new research

ideas and possibilities. The benefits and capabilities of the facilities increase when they inter-operate.

Federation shows other benefits a part from the capability to build a larger, heterogeneous system from multiple single systems belonging to different domains. The participation in a federation is a means to reduce the cost of infrastructure by the sharing of resources. When federated, a system can be used by a set of users much wider than the domain-specific set so the resources are more exploited and the social and economical global benefit is higher [12]. Federation is also a key concept for scalability of distributed systems and for building a composite distributed system or service federating single disconnected components [13]. Testbeds and other experimental facilities benefit from the advantages given by federation.

Similarly to the testbed scenario, federation also applies to the context of cloud. Multiple isolated clouds can be federated to compose a larger cloud, exposing the resources from each federated cloud as if they belonged to the same cloud. The federation might allow to improve the performance of individual clouds, optimizing the use of resources by using brokering agents. This idea is aligned with testbed federation. Actually testbeds can be seen as special case of clouds. A testbed, like a cloud, is a platform providing a pool of resources that are offered to the user as a service. While the cloud architecture offers different abstraction levels (Infrastructure as a Service, Platform as a Service, Software as a Service), a testbed can be seen as a special case of a cloud offering Experimentation as a Service (EaaS). In other words, a special cloud offering the service of defining and running experiments on a set of selected nodes. As a summary, federation is an important tool to leverage and improve the capabilities of facilities, specially in the FIRE scenario. Although being challenging, federation shows numerous benefits that help research and experimentation, and open a new range of possibilities.

2.3 CONFINE Community-Lab Testbed

Community-Lab (in short: C-Lab) [14] is a global facility for a experimentally-driven research in community networks. It is developed as part of CONFINE [15], a European research project that aims to provide a complete testbed platform for Future Internet Research and Experimentation (FIRE) infrastructures in Europe. C-Lab is a testbed for experimentation with network technologies and services in community networking. It provides researchers with tools to deploy, run, monitor and experiment with services, protocols and applications on real-world community IP networks. The motivation behind the testbed platform developed is to support and study the evolution of community networks in terms of their scalability and sustainability by providing the means to conduct experimentally driven research [1] [5].

The C-Lab testbed infrastructure is deployed on existing community networks [16]. Thus, the experiments performed do not run on a simulation of a community network but embedded on real community network. C-Lab consists of a set of testbed nodes connected to existing nodes of community networks,

so that they leverage the connectivity that the community nodes offer. The testbed nodes are connected among them and to the Internet by a community network.

The C-Lab testbed is currently in a production status, with 126 testbed nodes (2nd February 2014). Different existing community networks in production state participate in the testbed plugging in testbed nodes (research devices) to their community network nodes (community devices). These networks are AWMN (Athens, Greece), FunkFeuer (Wien, Austria), Guifi.net (Catalonia), Wireless België (Belgium), Ninux (Italy). Also different universities and research organizations participate in the project by maintaining testbed nodes. Universities such as UPC (Barcelona, Catalonia) [17], KTH (Stokcholm, Sweden) [18] and University of Tor Vergata (Rome, Italy) [19]; research organizations such as Pangea (Barcelona, Catalonia) [20], SICS (Sweden) [21], iMinds (Belgium) [22] and Unidata (Italy).

2.3.1 Testbed nodes

A C-Lab testbed node [23] consists mainly of a Research Device (RD) in which experiments and services are run. The RD is connected to a Community Device (CD) that is part of a Community Network (CN) and gives the RD access to the this community network. Thus the CD not only relays traffic related to the C-Lab testbed but also traffic of the CN unrelated with the testbed.

Keeping the research devices of the testbed (where the experiments are run) separate from the community devices that actually form the community network offers some advantages. On one hand, it avoids running experiments directly on CD and preserves their stability and thus, the stability of the CN. On the other hand, this separation facilitates the addition of testbed nodes to any CN and increases their compatibility with any CD. The addition of new testbed nodes consists of connecting a device to an existing CD with minimum or no C-Lab specific changes to its configuration.

Optionally, the RD can be connected to a recovery device. The purpose of the recovery device is to force remote reboot of the RD in case of malfunction, avoiding the need for hands-on access.

The mentioned devices (CD, RD and recovery device) are connected among them by a wired local network. In this network the CD acts a gateway for the RD, providing access to the CN and to the Internet. Multiple RD can be connected to the local network, all using the same CD as a gateway. The local network can also be shared with other non C-Lab devices like CN clients that use the CD as a gateway. Note that, strictly, the devices that form the testbed are the RDs, so the C-Lab testbed nodes are homogeneous. The Figure 1 shows the components and architecture of a Community-Lab testbed node.

2.3.2 Testbed management

The management of the testbed is done in a centralized way by one or several testbed servers. A testbed server is a normal computer whose minimum require-

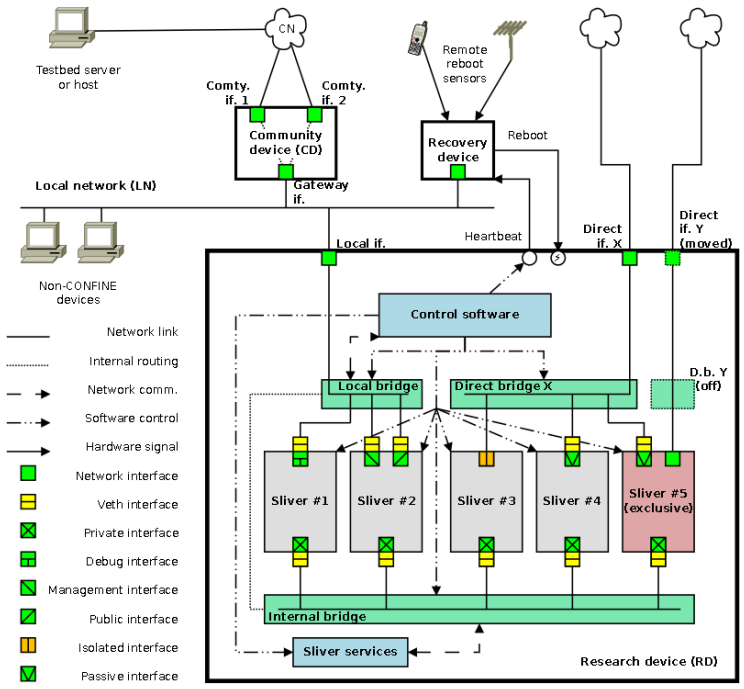

Figure 1: Community-Lab Node architecture (Source: CONFINE Wiki)

ment is to be directly accessible from the Community network and it acts as the controller of the testbed.

The testbed server provides access to a database (Registry) that contains the configuration for all the elements of the testbed. Moreover, it exposes the resources offered by the nodes that form the testbed. The resources are shared (and isolated) among users by using the concepts of slices and slivers from the Slice-based Facility Architecture (SFA), explained in detail on the next section. Briefly, the resources on each node are split into multiple portions (slivers) that are assigned to users. For running distributed experiments users can be assigned multiple slivers, grouped in a collection of slivers (slice). Reservation and release of resources are dynamic. In particular, reservation is not permanent and it automatically expires after some time.

In C-Lab three different actors can be identified. Testbed administrator is the actor in charge of managing testbed servers. Technicians are the actors who manage the nodes, which are owned by community members. Finally, testbed researchers are the users of the resources offered by the testbed. They define, run and manage the experiments in the testbed nodes.

2.3.3 REST API

The management of the testbed is based on a pull strategy in which components query the testbed registry to discover changes to their configuration and new action to perform. The pull strategy is implemented by a REST API [24] that servers and nodes expose. The REST API can be used by users to communicate with nodes and servers, but also by nodes that interact with servers to pull information. The Figure 2 illustrates the interactions explained in this section and the previous section.

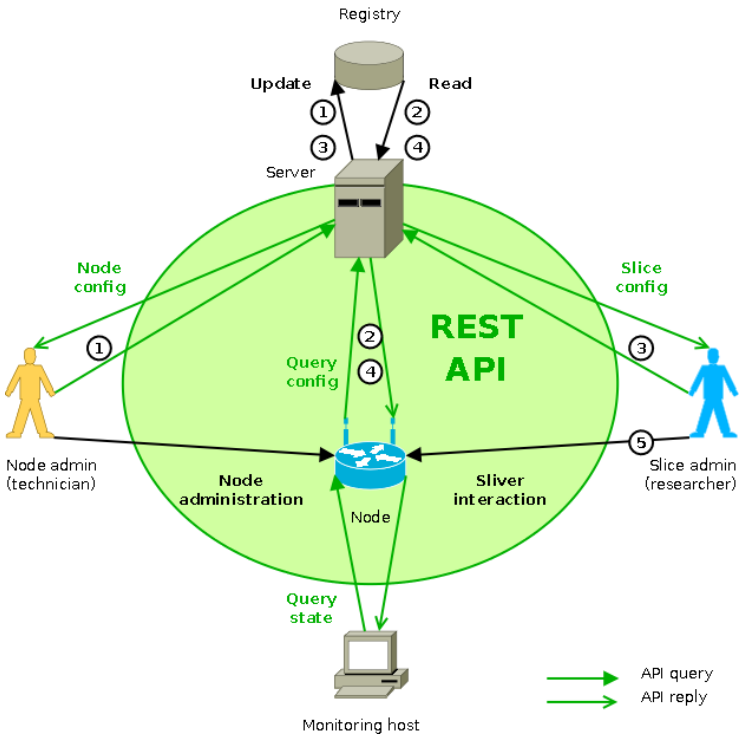

Figure 2: Interactions for Community-Lab testbed (Source: CONFINE Wiki)

In the following lines some implementation details of the REST API are presented:

- The REST API is implemented on top of HTTP secured with SSL.
- It is a navigable API. Resources include links to the API base, configurations and functions.
- Resources are described using JSON objects.
- Resources may include other resources. In such case, the description of the included resource does not contain the whole JSON description object but

a reduced version with a URI member from where the whole description can be retrieved.

- Lists of resources are represented as arrays of JSON objects. They allow member selection, filtering and combining.

2.3.4 CONFINE-ORM

Confine-ORM [25], which stands for Confine Object REST Mapper or Confine Object Resource Mapper, is a high-level Python library that exposes the REST API of the C-Lab testbed through a set of Python objects and methods that encapsulate the HTTP communication involved with the Confine REST API. Thus the library exposes high-level operations of the testbed such as create, update or delete on the nodes, slices and slivers. ORM is based on general design patters of the testbed REST API so it is able to navigate across the API and auto-discover the exposed objects and operations. Such feature gives to the ORM library an auto-maintenance behaviour.

Confine-ORM was initially developed as a tool for accelerating the testing of the testbed. However, due to its high-level functionality it becomes a client-side tool to easily interact with the Confine REST API and the Controller of the testbed through a programmatic interface.

2.3.5 Virtual CONFINE Testbed

The Virtual Confine Testbed (VCT) [26] is a virtual platform emulating the real Confine C-Lab testbed infrastructure in a local machine by using advanced virtualisation techniques. It provides an environment to quickly create a virtual network of CONFINE testbed nodes so a user can get familiar with the Confine software distribution, prepare experiments for real testbed and even extend real Confine networks with virtual links and nodes.

Besides the mentioned functionality for the end-user, an important purpose of VCT is to facilitate the development and testing of software and components for the C-Lab testbed. Such research and development tasks often compromise the stability and correct behaviour of the testbed. VCT offers a platform for these purposes while avoiding any damage to the real testbed. Moreover, VCT is a local platform, which facilitates the management, control and deployment operations throughout the research process.

2.3.6 Sandbox Controller

Besides the general CONFINE controller [27] that manages the real testbed and is publicly accessible, CONFINE also offers a sandbox controller for research and experimentation purposes. The idea of sandbox controller [28] is similar to VCT since the sandbox controller provides an environment in which code or content changes can be tested without affecting the production system [29]. It differs from VCT in that the environment provided is not local; the nodes managed by the sandbox controller must be real CONFINE nodes, like for the real controller.

The sandbox controller is publicly available as well but it only manages very few nodes.

2.4 Experimental testbed federation and Fed4FIRE Project

Federation of experimental testbeds is seen nowadays as a good activity to exploit the capabilities of such facilities and extend the range of possibilities for the Future Internet Research and Experimentation (FIRE). Therefore, many projects focus their efforts on the federation task; Fed4FIRE is one of this projects.

Fed4FIRE is an Integrating Project under the European Union's Seventh Framework Programme (FP7) [30]. Fed4FIRE, which stands for "Federation for Future Internet Research and Experimentation", aims to implement a federation framework for a set of European facilities for research and experimentation. The facilities willing to be federated are heterogeneous and address different domains within the Future Internet ecosystem. The Community-Lab testbed is one of these FIRE European facilities. Thus, the federation tasks developed in this thesis are a part of the Community-Lab contribution to the Fed4FIRE project.

2.5 SFA

The Slice-based Facility Architecture (SFA) is an architecture specification for testbeds that standardizes the sharing of the resources of a testbed and facilitates the federation with other testbeds [31].

SFA provides a proper framework that enables key operations on the pool of resources offered by the testbed. Basically, SFA allows to perform resource allocation, resource management, resource sharing and resource usage (for running experiments). For such purposes SFA defines different abstractions, managers and principals [32]. The Figure 3 illustrates such elements and the interaction among them.

Abstractions

Testbeds consist of a set of nodes that provide resources. SFA uses the concepts of components, slivers and slices to handle those resources and share them among researchers:

- Component, which represents the minimal aggregation of physical resources that can be managed.
- Sliver, which is the portion of such resources let to a researcher.
- Slice, which is a collection of slivers assigned to a researcher to perform an experiment.

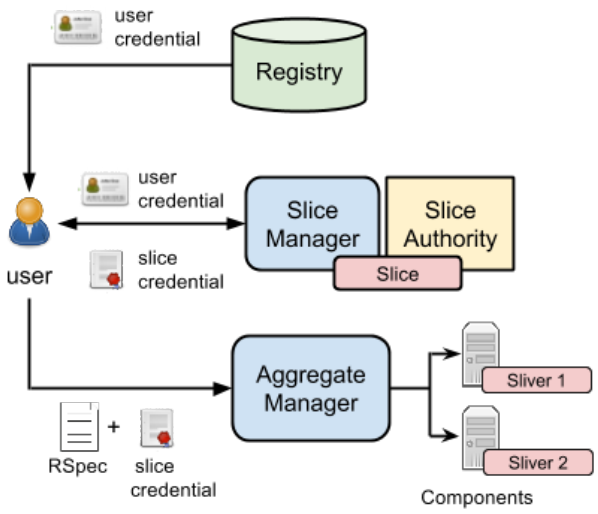

Figure 3: SFA elements and their interaction (Source: own creation)

Slices can be seen as containers of slivers used by users to group a set of slivers in which a specific experiment is run. Therefore, slices become the primary abstraction for accounting and accountability. The life-cycle of a slice consists of three stages that must be followed in order: (i) register: the slice exists only as a name bound to a set of users; (ii) instantiate: the slice is instantiated in the required components, being granted of a set of resources; and (iii) activate: the slice becomes active and runs code on behalf of the researcher.

Managers

The defined abstractions are managed by a set of managers. The managers expose a specific interface that allows the control of the resources that they are responsible for. Depending on the abstractions managed three different managers are defined:

- Component Manager (CM), manages single component abstractions.
- Aggregate Manager (AM), manages a collection of components that behaves as a single aggregate component.
- Slice Manager (SM), enables the management of slices and slivers, acting as a proxy to the corresponding Aggregate Manager responsible for the resources of the slice or sliver.

Besides the mentioned managers there is another important component in the architecture of SFA, the Registry (R). The Registry keeps track of all the object abstractions of the testbed storing them in a database as SFA records. The Registry can be read and modified through its interface, that allow CRUD operations [33] on the SFA records. It also handles permissions for the testbed objects by the usage of credentials. The Registry is responsible for issuing and checking credentials to operate on the testbed objects.

Principals

Principals refer to the the different actors or roles that can be identified on SFA. Among the roles defined by the principals there are the authorities. Authorities represent testbeds, part of testbeds or communities of users. The objects abstractions of SFA are registered in the context of an authority that is responsible for their correct behaviour. In SFA three principals are defined:

- Management Authority (MA), responsible for a set of physical components and ensuring specified allocation policies.
- Slice Authority (SA), responsible for slices, their access and their control.
- User, a person that plays one or more roles in a facility.

Some example of users are: a researcher user that runs an experiment or service in a slice, an operator of the testbed that manages a part of the physical resources, or a client that uses the services deployed in slices.

Federation

The SFA architecture also helps to provide a standard interface for federation. Different testbeds following the SFA specification can be easily federated to work together. The goal is to provide a minimal interface, a narrow waist, that enables testbeds of different technologies and/or belonging to different administrative domains to interoperate without losing control of their resources [34].

2.5.1 RSpec data type

RSpec is an XML document that describes physical resources. Any SFA operation that needs a description of physical resources uses an RSpec document. An RSpec documents follows a standard schema defined by the GENI [35] initiative. The RSpec document has three different purposes and therefore, and GENI defined three distinct schemes with small language differences to address these three specific purposes:

- Advertisements, used to describe or advertise the available resources on a specific Component Manager or Aggregate Manager. The Advertisement RSpec describes the characteristics that resources offer in order for the clients to choose which resources are more suitable for their purposes.

- Requests, used by a client to specify which resources are selected. The Request RSpec does not need to contain detailed information about the selected resources. It only needs information to identify the desired resources and perform the mapping between physical resources and testbed objects.
- Manifests, used to provide information about the resources being used in the testbed. A Manifest RSpec is used by the wrapper itself in some replies, when it is necessary to describe the slivers that are being used by a client.

There are different versions of the RSpec XML Schemas defined by GENI. The versions that are currently in use are version 2 [36] and version 3 [37].

2.5.2 Credentials

A credential is an XML document that describes the privileges of an owner on a target. Normally the owner of a credential is a user and the target for which the owner has privileges is usually a slice or sliver. Both owner and target of the credential are unambiguously identified by a using unique identifiers. The credential is issued by the Registry and it is signed by an authority to guarantee its authenticity. The credential has an expiration date after which it is not valid anymore.

There are different types of credentials supported in SFA: GENI SFA version 2 [38], GENI SFA version 3 [39] and GENI ABAC [40].

2.5.3 URN and HRN

Two different unique identifiers are used to unambiguously identify objects of the testbed: Uniform Resource Name (URN) and Human Readable Names (HRN).

The URN is a uniform resource identifier used in computing that follows a standard scheme. In particular, URNs in SFA have the format:

```
urn:publicid:IDN+<authority string>+<type>+<name>
```

where type identifies the type of object or record (authority, user, slice, sliver, node).

The HRN is a specific identifier of SFA. As the name suggests, it is easily readable by humans, in contrast with the URNs. The HRN of an object allows to identify the sequence of authorities that are responsible for the object. This hierarchy results in HRNs with the format:

```
top_level_authority.sub_authority.sub_authority.name
```

2.6 SFAWrap

SFAWrap is a free software that allows to federate a testbed into the emerging SFA-based global federation of testbeds. The software package provides a set of

general components to help any testbed to expose a SFA-compliant interface, which integrate the generic part of the code. It also includes a set of skeleton classes to be implemented for each particular testbed, which integrate the testbed-specific part.

2.6.1 Generic part

The generic part of the code provides a set of packages and modules that implement general and not-testbed-specific components and elements of the wrapper. The generic code implements all the standard processing related to SFA that does not depend on the testbed. Basically, what is included in the generic part of the code is:

- **SFA interfaces and Servers**. The generic part of the code exposes the SFA standard interfaces that receive SFA calls from SFA clients. All these calls follow the standards of SFA so they are completely generic and no testbed-dependent. The interfaces are exposed through a set of XML-RPC servers implemented as multi-threading servers.
- **Authentication and authorization**. The generic part of the code handles the authentication and authorization operations performed in SFA. The management of credentials as well as their creation is performed by the generic code. The mechanisms used are typical of SFA and independent of the testbed. Therefore, when a SFA call is received by a server, the generic code performs all the checks on the mandatory credentials presented by the caller. In other words, it checks that the credential presented shows that the caller has the necessary privileged to perform the requested operation.
- **Generic Registry**. A generic SFA registry is also implemented following the standards of SFA in terms of records, formats and identifiers. The registry is queried and/or updated in most of the operations, so it represents all the objects of the testbed created through the SFA interface.
- **GENI RSpec management**. The generic part of the code includes modules to manage the standard GENI RSpecs, version 2 and version 3. The module provides methods to correctly create and manipulate the RSpec documents. However, each testbed can define its own version of RSpec including specific elements to represent characteristics of the testbed. In this case it is necessary to define new modules and classes to manipulate the new RSpec version, similarly to the generic modules that manage the GENI RSpecs.
- **Client tools**. Two client tools are also included in the generic code. The client tools are implemented as command-line tools that allow to configure the SFA endpoint and send SFA calls to it.

2.6.2 Testbed-specific part

In addition to the generic part of the code, SFAWrap has a testbed-specific part. Basically, this part of the code acts as a "glue" between the generic part of the wrapper and the testbed. It is in charge of translating the API of the specific testbed and adapt it to the SFA operations as the wrapper requires. The testbed-specific part is what is really needed to implement a SFAWraper for a new testbed. Basically, the testbed-specific part consists of a driver that exposes the standard SFA operations that the generic part requires, but using the API of the testbed to correctly implement the operations. The driver includes several modules with functionality that will be explained in more detail in the next paragraphs.

Dummy testbed

In order to make easier the task of implementing the testbed-specific part of the wrapper, the code includes an implementation example of the driver for a Dummy testbed. The Dummy testbed is a fake testbed implemented as a Python module and it provides a typical testbed API. The testbed API is used to implement the driver for the Dummy testbed, so that the SFAWrapper can be configured to operate with the Dummy testbed. The testbed is fake as it only provides an API with the empty methods that reply correctly, but they do not have any effect in terms of resource reservation. The purpose of the Dummy testbed is to show how the wrapper works and what is needed to implement and configure a driver for a testbed. The effects that operations have on the testbed can be seen by querying the registry that stores all the records of the testbed.

The Dummy driver is an example of implementation and it can actually be used as a template for new testbeds. Dummy driver includes all the necessary modules and methods that are required to implement a SFA wrapper for a testbed. It is a guide that a developer can use to understand which modules integrate the driver and what the methods in these modules are and what their tasks are.

Implemented wrappers for other testbeds

In addition to the Dummy testbed example, the SFAWrap code also includes other examples of implemented drivers for real testbeds. The drivers, together with the SFAWrap and the generic code provide an SFA interface for the testbeds. The drivers are ready to be used so the SFAWrapper can be configured to operate with one of these drivers and interact with a real instance of a testbed through the SFA interface that the wrapper provides. The different drivers that SFAWrap includes that can be configured to be used are called *flavours*. To configure a flavour in SFeAWrap some configuration parameters are needed. These parameters depend on each testbed and driver, and they have default parameters for the included flavours.

The flavours included in the SFAWrap package code are:

- CorteXlab [41], an heterogeneous radio testbed to evaluate different aspects of cognitive radio in real environment.
- Federica [42], an experimental network infrastructure for trialing new networking technologies.
- IoTlab [43], a testbed infrastructure for multidisciplinary experiments with more end-user interactions by extending the testbed with the potential of crowdsourcing.
- Nitos [44], a wireless experimental testbed that is designed to evaluate networking protocols and applications.
- OpenStack [45], testbeds based on the OpenStack Open Source Cloud Computing Software to build a Cloud.
- PlanetLab [46], a network of open computers distributed across the world and available for the development of new network services.

Add a new flavour

For using SFAWrap with a new testbed is necessary to add a new flavour. The task of adding a flavour consists of implementing the driver for the testbed and adding the required configuration to allow the wrapper to use the driver.

The implementation of the driver basically implies to wrap the methods offered by the testbed API in such a way that the required SFA method behave as expected on the testbed.

The configuration needed for a flavour depends on each testbed and the parameters that a testbed required for usage. Usually, the configuration for a testbed includes information about the endpoint or URL where the testbed API is exposed and the account information of the user. Some testbeds also include specific options or configuration parameters that apply in their specific domain.

The configuration for each flavour can be set at the moment of selecting the flavour and initiating the wrapper. However, a default configuration is specified and usually the user can keep it for a correct usage of the wrapper. The default configuration is stored in a XML file. This XML file must be modified for each new flavour that is willing to be added in the SFAWrap.

3 Federation in Community-Lab testbed

This chapter discusses the testbed federation scenario. Section 3.1 discusses the different domains and possibilities in such general scenario. Sections 3.2, 3.3 and 3.4 focuses on the particular testbed Community-Lab, identifying concrete federation scenarios for this testbed. Namely, multiple controllers federation in 3.1, federation with WiBed testbed in 3.2 and federation with other external facilities in 3.4.

3.1 Domains and possibilities

In a federation scenario among heterogeneous facilities (e.g. testbeds) there is a trade-off between the level of homogeneity achieved by the federation and the ability to leverage the particular features of the facilities. This trade-off is related to the desired level of federation among the facilities. In other words, what constitutes the inter-interoperability provided by the federation.

From the point of view of a user, federation might provide a unified single interface to interact with all the federated facilities, rather than using a specific interface for each facility. In such unified interface the process of interacting with the facilities is the same for all of them, which means that the heterogeneity of the different facilities must be absorbed by this process. In terms of testbeds, the typical use case basically consists of discovering available resources, allocating and instantiating resources and using the resources by deploying and running experiments on them. Depending on the nature of the testbeds and their architecture and native resource model this unification task can be difficult or even infeasible.

Still talking from the user side, the facilities usually require to create an account to use them, granting a set of permissions for the user. The unification of the interface might somehow imply an accounting federation as well. After a user logs in he should be able to interact with the federated facilities, perhaps having different permissions on them. Federation usually consists of allowing the interoperability of existing facilities without these facilities losing any control or previous functionality. For authentication and authorization, each facility will keep working with its own accounts (and interface if applicable) and with the accounts from the federation domain, either from a central registry for the federated facilities or from multiple trusted domains.

The final end of the federation is to enable the combination of functionality, features and capabilities offered by the different federated facilities for allowing more complete, cross-domain research. In other words, design experiments using a combination of heterogeneous resources that can inter-operate, so that the aggregate of their offered features can be exploited. A wide federation can include many facilities so it might be difficult to know for a user which features are provided by each of them. Instead, the user might specify which are the requirements in terms of resources that are needed for the experiment to be performed. The mapping between this set of requirements and the resources reserved in the different federated facilities can be done by a broker agent. Apart

from a criteria about functionality, the broker can also take into account metrics of cost, load or performance to decide the most suitable facility to allocate the resources. The broker acts as a federation central agent in this scenario because, carrying out the federation-related tasks automatically on behalf of the user.

Federation implies a standardization for example in terms of the process of resource reservation. Besides that process, standardization can be extended to other levels, such as the definition of experiments, or orchestration, instrumentation and monitoring of experiments. The process of designing and deploying an experiment on a set of reserved resources may depend on the particular facility and its process for this task. When combining resources that could belong to different heterogeneous facilities in the same experiment, the need for standardizing this process becomes evident. An equivalent scenario can be identified for the instrumentation of the resources and the monitoring of the experiment. Then, general tools for carrying out such tasks in aggregate of federated facilities are needed. In general, having such a unified interface allows the usage of generic (non-testbed specific) tools to interact with the testbed.

The idea of brokering can also be extend to the domains mentioned on the previous paragraph. In addition to the description of resources, a broker may receive the experiment description, so it can deploy the experiment on the resources after the reservation process is finished. The deployment process could even vary depending on the facility containing the resource and the broker could hide this heterogeneity to the user. However, the lack of standards for all these processes might lead to sustainability and scalability problems if the broker (or the federation entity) needs to deal differently with each federated facility. Therefore, as pointed out previously in this section, federation is closely related to the standardization of processes and models used in the underlying facilities.

Federation implies the interoperability among different domains and therefore, the concept of transparency is also important. It addresses questions like "does the user needs to be aware of the different federated domains?" or "what the user needs to know to about the federated domains to work with them?". By definition, federation implies a certain level of transparency enabling such interoperability. Moreover, the level of transparency can be total if the federation entity behaves as a single entity (although being integrated by an aggregate of entities) from the point of view of the user. On the other hand, this transparency might be reduced due to, for example, issues related to the underlying communication among the facilities. Resources in the federated facilities will need to communicate with each other, which means that they must be reachable. However, facilities can use networking solutions, such as NATs or VPNs, which avoid these publicly-reachable feature. Particular steps are then needed to set up such communication, thus reducing the level of transparency from a user point of view.

In summary, there can be different domains or levels of abstraction in which federation applies. The level of transparency that the user perceives depends on the the domains in which federation among the facilities is carried out. Interface, accountability, brokering, resource sharing or networking are different domains

in which federation applies.

Next sections bring some of the ideas discussed along this section to the context of C-Lab testbed, identifying some possible federation scenarios for this particular testbed.

3.2 C-Lab Controllers federation

Currently the C-Lab testbed uses a single controller that manages the resources of the testbed and exposes the API of a single, centralized registry that keeps track of all the objects in the testbed and their configuration. The controller modifies the registry according to the user operations performed; modifications in the configuration of objects are eventually applied since the objects themselves query the registry through the controller API following a pull style.

The single registry (and its controller) manages all the nodes (resources) of the testbed, which are distributed across Europe: Barcelona (Spain), Antwerp (Belgium), Wien (Austria), Athens (Greece), and Rome (Italy). More controllers could be added (for example one for each region) so that the resource control is distributed and the load of the single controller is split up among them. Other advantages of the scenario with multiple controllers are that the network latency from the nodes to the controller will decrease and that the controllers can allow different policies in the different regions.

Controllers then would need to be federated to allow the interoperability among them. Such federation scenario corresponds, in fact, to the distribution of the single Registry. There will be several distributed instances of the registry rather than a single, centralized one, with each instance taking care of a subset of testbed objects and resources.

The federation among the controllers can be done using different approaches. Note that in this case we are federating homogeneous testbeds, i.e. federating controller testbeds that expose exactly the same API. Therefore, no new standard interface or layer needs to be implemented, but using the existing API to enable the interoperability among different instances of the controller.

API extension with References to federated controllers

One of the options for carrying out the federation is that each controller stores references (links or pointers) to the other federated controllers and uses these references to interact with them. Extending the controller API, such references may be stored in a (now non-existing) federation directory. Thus, the controller can (automatically or by explicit request from the user) use the references for contacting the federated controllers in any operation. The references to the controllers will logically correspond to the base URI of each controller, from which all the API is reachable by simple navigation.

For example, imagine a federation system with federated controllers A, B and C. A user invokes a `node list` operation on the controller A. The controller A will list its own nodes, showing the information from the node API; then the controller will get the base URIs of the federated controllers form the federation

directory; using these links, the controller A can retrieve a list of node URIs from the controller B and the controller C, or even the information of the node API can be retrieved. The final result for the `node list` operation on controller A would be either a list of nodes that contains the API information of nodes from controller A, B and C or a mixed list with API information of nodes from controller A and a list of node URIs from controller B, C (from which node API information can be retrieved). The idea is the same when using (instead of listing) a node from a federated controller. The operations will be invoked on a remote URI rather than a local one, so the operation will be handled by the federated controller.

The approach of using references to federated objects offers a certain level of transparency. Such transparency can be total when allowing the controllers to perform recursive queries on the federated controllers through their references, so the raw API information can be retrieved and showed as if it was local. However there is an important drawback on such approach: the need to modify/extend the controller API. The API needs to accommodate modifications to deal with the federation directory and the references to the federated controllers. Conceptually that would mean that for designing a new service, such as federation, the core of the system is being modified. This does not seem a good practice, besides the obvious sustainability problems of adapting all the existing services to the required modifications. Instead, keeping the core of the system as simple as possible and moving the complexity to higher-layers seems a more reasonable choice (and already adopted in many areas of computer science like Microkernel [47] and the Web [48]). In terms of the testbed, the controller API should not be modified to implement the new federation service.

Higher-level federation service with Proxy controllers

In this approach the idea is to keep the controller API as simple as possible and build the new services, such as federation, as higher-level components or applications. The federation is implemented besides the controller API as a user service. The service maintains the federation directory with the base URIs of the federated controllers. Unlike the previous approach, the controller behaves as a single, isolated controller without knowing anything about federation, and it is the user by explicit usage of the federation service who enables the federation. The user selects when performing a certain operation to a federated controller. The controller API does not need any modification and it remains simple, at the cost of losing transparency on the federation scenario. In this scenario the user is federation-aware.

Although being a more simple approach, there are some open issues that must be considered. The federation service implemented will allow the user to invoke certain operations on remote controllers. Thus, such operations are performed by logged users that belong to groups and are associated with given slices. Both the user and the slice are objects that belong to the local controller and are stored in the registry for which the local controller exposes the API. A federated controller does not know anything about the content of other federated

registries. Therefore, the federated controller will not know the mentioned user and slice objects from the local controller. This raises a problem of how to perform the operations on the federated controllers; to which user and slice the operations are associated with. The users can be validated by a chain of trust, so other controllers recognize them as legitimate, valid users. For other objects, like slices, the current API uses URIs as references. However, in this scenario such URIs could reffer to non-local objects rather that only local ones. This would imply the need of modifying the implementation of the controller (the API might not need changes) to handle such external URIs.

In this federation scenario, a controller will therefore handle the local requests coming from its own users, and the requests coming from external users.

To optimize the traffic and reduce the load of each controller a Proxy controller can be used. The Proxy controller is a replicated instance of the controller that receives the requests. It is provided with a cache system that stores the results of previous read-only operations. Thus the Proxy can immediately reply to the read-only/no-modification operations with the information from the cache. Periodically the information of the cache expires and the Proxy will forward the call to the real controller. The eventually-consistent model that is used in the Community-Lab testbed matches with the consistency of this Proxy approach.

The usage of a Proxy controller could also help reducing the load of the controller for write/update operations. Leveraging again the eventually-consistent model, the Proxy controller could immediately acknowledge the write/update operations, and queue the operation requests to be sent later to the controller. The Proxy could wait for having a batch of (related) operations before sending them to the real controller; if the operations consists of accessing a database and performing some modifications, such behaviour could optimize the tasks. The modifications would be queried by the objects of the testbed following the pull strategy and the changes would be eventually applied. This modification would need further study to evaluate the potential improvement of the performance and the impact on the changes-application time.

3.3 C-Lab and WiBed

In C-Lab node virtualisation techniques are used to allow running different experiments in parallel on the same node. However, the adoption of virtualisation has two main drawbacks: (1) higher node costs due to increased requirements on computing resources, which might reduce the number of nodes being deployed; (2) restricting access to lower layers of operating system and communication stack due to the need of ensuring isolation of experimentation resources, which restrains the range of supported experiments [49]. To overcome such drawbacks the testbed platform Wibed was developed (without using virtualisation techniques) as a complement to the C-Lab testbed.

Wibed [50] is a platform for facilitating the deployment and management of testbeds based on commodity IEEE802.11 routers, which enables the experimentation with wireless technology including the modification of low-level system components (physical and link layer mechanisms, network and transport

layer protocols). The Wibed platform has been used to deploy the UPC CN-A testbed that consists of 50 nodes over six buildings of Universitat Politècnica de Catalunya (UPC) Campus Nord, Barcelona. UPC CN-A testbed is willing to be federated with Community-Lab testbed. This way, C-Lab can increase the range of supported experiments by leveraging the advantages of accessing low-level communication layers in UPC CN-A testbed nodes.

Although C-Lab and Wibed UPC CN-A testbeds are different in terms of architecture and offered functionality, the design efforts in the Wibed API are in the direction of developing a (maybe reduced) API equivalent to C-Lab API. It means that Wibed-based testbeds will expose an API (or at least a subset of) compatible with C-Lab API. In that sense, Wibed specific concepts will need to be mapped into SFA concepts. The federation scenario between Wibed testbed and C-Lab testbed is similar to the multiple C-Lab controller federation scenario. Basically, the idea is again to federate two testbeds (or the controllers of the testbeds) that expose an homogeneous API.

As analysed in previous section, the approach of Higher-level federation service based on Proxy controllers might be a suitable approach for such task. Assuming that the Wibed Controller API is equivalent to the Community-Lab Controller API and it is also compliant with the SFA model, the Wibed controller can be included in the federation service, and used as a federated controller.

3.4 C-Lab federation with other facilities

Extending the federation to a wider scenario, C-Lab can be federated with other heterogeneous facilities, i.e. facilities based on other architectures, models and with different APIs. In the previous sections a particular solution for federating multiple C-Lab controllers and Wibed controllers was discussed, leveraging the homogeneity of the API and the architecture. However, in the current scenario the heterogeneity of the facilities being federated reveals the need of a standardization in the federation, for complexity and scalability reasons. The idea is to establish a standard model for federation so that the facilities compliant with such model can be easily federated; that is, follow an architecture model and expose a standard API.

SFA has been adopted as the federation standard for testbeds. If the testbeds provide somehow a SFA-compliant interface, the scenario turns into a scenario with homogeneous testbeds and the same federation solution works for all them. The SFA interface can be implemented as layer that adapts and maps the specific testbed to SFA. The implementation of such layer for C-Lab is the final goal of this thesis.

In this general scenario, the federation between C-Lab and Wibed can be seen in two different ways. On one hand, C-Lab and Wibed can be seen as different testbeds, so each one exposes its SFA interface on top of its corresponding API. In this case, this general federation solution (SFA interface) could even be used for the particular federation between the two testbeds. On the other hand, the Wibed testbed can be seen as a part of the C-Lab testbed, which can

be achieved by using the particular federation solution on previous section. In this case, only the C-Lab controller will expose the SFA interface. The C-Lab controller acts as an aggregate of testbeds and the federation between C-Lab and Wibed is transparent to SFA users.

4 Design

The problem to be solved is the federation of C-Lab with other testbeds and FIRE facilities. Such goal is accomplished by designing a software tool to be used as a complement for the C-Lab testbed to enable the federation. The design of this federation tool is exposed in this chapter. Section 4.1 presents the solution adopted, that is, using SFA for federation. Section 4.2 explains how this model is applied to Community-Lab. Sections 4.3 and 4.4 describe the federation tool implemented and its architecture respectively. Finally, the most important design decisions are presented in section 4.5.

4.1 Federation solution adopted

The study of the different solutions for federating the testbeds, and in particular for the C-Lab testbed, was out of the scope of this thesis. The federation of the C-Lab testbed is a part of a larger European project whose goal is to build a global infrastructure of experimental facilities, such as testbeds, for Future Internet Research and Experimentation (FIRE). At this end, a standard for interaction and interoperability among the facilities is needed as a federation solution. A survey of the different possibilities to federate the facilities was performed as a first step in the Fed4FIRE [13] and CONFINE [51] European projects.

. In the survey [13] the architectural requirements for the federation solution were identified and based on that four different architectural approaches considered the most relevant architectural candidates were evaluated. It was concluded that *Heterogeneous federation* was the most suitable approach. The solution consists of an heterogeneous federation where all testbeds run their native testbed management software, but common, standardized interfaces run on top such testbed management software to enable federation. Mutual understanding and interoperability are thus achieved through these standardized interfaces.

The next step in the context of the Fed4FIRE project once decided the architectural model was to define the standardized interface. In other words, described the API exposed by the common interfaces through which testbeds can interoperate and be federated. The experience of PlanetLab project [46] was a valuable point of reference for such task.

PlanetLab is a group of computers available as a testbed for computer networking and distributed systems research [52]. The project started in US in 2002 and the initial deployment consisted of 100 nodes at 42 sites across the country, with all sites governed and managed by the PlanetLab Central [53] authority. The extension of the project to Europe started in 2004, deploying new nodes at sites across Europe managed by their own authority PlanetLab Europe [54]. Later in 2007, OneLab project [55] put its effort on federating the different PlanetLab authorities to build a global, worldwide testbed platform. In order to achieve such goal, *Slice-based Facility Architecture (SFA)* [56] approach was adopted. Nowadays, PlanetLab testbed consists of 1188 nodes at

582 sites worldwide (April 2014), managed by federated PlanetLab Central and PlanetLab Europe authorities.

The PlanetLab federation model was taken as a reference for federation of facilities and testbeds in the European project that serves as a context for this thesis, so the the common interface that the facilities need to provide on top their management software follow the SFA specification.

Summarizing, the federation solution adopted for federating C-Lab with other testbeds was not a design decision but a requirement imposed by the European project. It consists of an heterogeneous federation model in which testbeds provide a standardized SFA interface on top of their management software to enable interoperability among them. The SFA interface is developed as a software layer that acts a "translator" between SFA and testbed-specific domain.

4.2 SFA in C-Lab testbed

The architecture of C-Lab testbed is highly based on SFA [57], since it has become the most important and standard specification for testbed architecture and federation due to the influence of PlanetLab. Thus C-Lab testbed uses the abstractions defined by SFA: components, slices and slivers.

In C-Lab the components are nodes implemented by Research Devices. The nodes that conform the testbed are homogeneous and all have the same characteristics. The idea of homogeneity is also extended to slivers. Slivers are the portions of the resources that are assigned to a user. Such portions are homogeneous and are implemented as virtual machines inside the nodes conforming the testbed. In both abstractions nodes and slivers, the hardware resources are not customizable and they are pre-defined. It means that all the slivers that a user can request are equal in terms of hardware characteristics. The customizable characteristics are reduced to the set of network interfaces available in the sliver. In terms of software characteristics the user can define the image of the OS to be used in the sliver.

C-Lab also supports the concept of slices. Like in SFA, slices are the primary abstraction for accounting and accountability. A slice is a collection of slivers assigned to a researcher and it represents the set of resources borrowed to a user to perform an experiment. The resources, that is slivers belonging to slices, are assigned and released dynamically. To control such dynamism slices and slivers have a state parameter.

The slice concept in SFA is simpler and lighter than the one in C-Lab. While in SFA a slice is seen as container for slivers that does not support any kind of configuration, in C-Lab there are some configurable characteristics of the slice that when set, are inherited by all the slivers belonging to that slice.

Another important difference between SFA and C-Lab is how they manage the nodes, slices and slivers abstractions. In SFA there are different manager modules for that purpose. Component Manager is the module that manages a component or node. When the same component manager manages multiple components it is called Aggregate Manager. The slices and slivers are managed

by the Slice Manager. In C-Lab the management of nodes, slices and slivers is centralized and it is carried out by a single component, the controller of the testbed.

In SFA there is also a Registry entity that keeps track of the different objects of the testbed, such as slices, slivers and users, assigning a unique global identifier to each object. Such database also exists in C-Lab testbed and it keeps the state and configuration of all the objects. Periodically the objects retrieve their configuration and perform the necessary actions, thus following a *pull* strategy. The Registry database in C-Lab is maintained by the controller of the testbed. There is another important abstraction in SFA that the Registry keeps track of and that is not directly mapped into any abstraction of C-Lab: the authorities.

Authorities represent testbeds, parts of testbeds or communities of users. These authorities are in charge of the objects registered with them and are the entities that take care of authentication and authorization for the usage of such objects. SFA uses certificates and credentials to grant rights to a particular principal or user. These certificates are issued by a trusted authority and are manipulated by the managers.

In C-Lab the notion of authority does not exist. Instead, C-Lab defines the concept of group. Each abstraction among nodes, slices, slivers belong to a group of users. The users that are part of a group are possible candidates to have rights over the resources that belong to this group. At the same time, a user can be a member of multiple groups, with different permissions and rights in each group.

The mapping between concepts and abstractions of C-Lab and SFA is quite direct and easy in most cases, due to the fact that C-Lab bases it model on SFA. However, specific entities or concepts from testbeds that are not SFA-based might be difficult to map into SFA standard concepts. Table 1 shows a summary of the mapping between concepts from SFA domain and C-Lab domain.

4.3 SFA Software layer: SFAWrap

Different options for implementing the SFA software layer were discussed in the context of the European project. Each testbed partner selects the most suitable option depending on the its characteristics and architecture. Basically the options are divided in to branches:

(a) Implement the SFA software layer using the SFAWrap tool

(b) Implement an own, specific solution. For example, implement a software SFA-translation layer from scratch, implement directly a SFA-compliant testbed, or any hybrid solution between them

For C-Lab the option selected was the usage of SFAWrap. The reasons for making such decision are:

- C-Lab testbed was fully implemented and developed so the option of directly implementing a SFA-compliant testbed was discarded

SFA	C-Lab	Comments
Components	Nodes	Nodes implemented as Research Devices. Homogeneous, different network domains
Slivers	Slivers	Homogeneous. Implemented as VMs in nodes. Customizable network interfaces and OS image
Slices	Slices	C-Lab slice is heavier entity: configurable characteristics inherited by its slivers
Component Manager	Controller	Centralized controller manages Nodes
Aggregate Manager	Controller	Centralized controller manages Nodes
Slice Manager	Controller	Centralized controller manages Slices and Slivers
Registry	Controller	Controller keeps a database of testbed objects
Authority	Group	Authority concept does not exist in C-Lab. Instead, *group* is used to define ownership for nodes, slices and slivers

Table 1: Mapping between SFA and C-Lab concepts

- C-Lab architecture is highly based on SFA model which makes feasible the adaption of testbed-specific concepts (entities, objects, functionality...) to SFA domain
- Existence of Confine-ORM library, a Python library that encapsulates the REST API of C-Lab testbed. SFAWrap is also written in Python and it requires to implement the SFA standard methods for the testbed being adapted. Therefore, ORM is extremely suitable and useful for such task
- SFAWrap already provides generic features and functionality related to SFA (XML-RPC servers, credentials and authority management, support for Registry)
- SFAWrap offers integrated command-line client tools that can be used for testing during the development process
- SFAWrap includes some examples of existing wrappers for other testbeds, as well as a Dummy testbed example that can be use as a guide

The reasons listed above revealed the *a priori* feasibility of using SFAWrap for C-Lab. The fact that C-Lab testbed is based on SFA model allows the pos-

sibility of using SFAWrap; for non SFA-based testbeds the translation between testbed-specific and SFA domain might be more complex and designing and ad-hoc wrapper from scratch might be easier. If possible, using SFAWrap seems to be a better option since generic features offered by the package can be leveraged instead of implementing them from scratch.

4.4 Architecture

The development of SFAWrap for C-Lab started with the study of the Dummy Driver that, as previously said, can be used as a template for new drivers, in terms of architecture, modules and methods. It is an implementation of SFAWrap for a fake Dummy testbed. The testbed is simulated through a `dummy_testbed_api.py` module that acts as a server for the testbed, exposing an example interface that in fact does nothing but replying successfully to the received calls without performing any action in terms of resource allocation or provision.

The Dummy Driver consists of five Python modules, each module in charge of a different functionality in the wrapper. These modules are: `dummy_driver.py`, `dummy_aggregate.py`, `dummy_slices.py`, `dummy_shell.py`, and `dummy_xrn.py`. The implemented driver for C-Lab follows the same idea, separating the modules by functionality, even with a higher granularity than Dummy driver.

The modules and their corresponding functionality for C-Lab driver are the following:

- `clab_driver.py`, driver module with the operations that the SFAWrap driver requires. Its methods encapsulate calls to other `clab_*` modules.
- `clab_aggregate.py`, module that exposes the Aggregate Manager API (GENI AM API v3 [58]) for the C-Lab testbed.
- `clab_registry.py`, module that exposes the SFA Registry API for the C-Lab testbed.
- `clab_slices.py`, auxiliary module that checks the nodes and slices in the operations of the AM.
- `clab_shell.py`, module that exposes the API of the testbed through XML-RPC calls to the REST API of C-Lab testbed. It uses the ORM library as a programmatic interface for accessing the REST API.
- `clab_xrn.py`, auxiliary module that handles the translation between URN and HRN names to testbed-specific names and viceversa.
- `clab_exceptions.py`, module that defines the possible testbed-specific exceptions handled by the driver.
- `clab_logging.py`, auxiliary module to generate log files for the driver.

In addition to the mentioned modules, that correspond specifically to the modules that integrate the driver of SFAWrap, there are two more modules needed for a complete and correct behaviour of the wrapper:

- `clab_importer.py`, module that reads the database of the testbed and imports all the elements to the SFA Registry database, properly translating the information into SFA records.
- `clab.py`, generic module that specifies which are the modules and methods of the driver that the generic part of the code has to call for each operation when using the C-Lab flavour.

The modules listed above implement the structure of the driver. The granularity and division used is based on the Dummy driver example. However, there are some differences since the Dummy driver was an extremely simple example and the driver for C-Lab required more content. The Dummy driver only includes the modules `dummy_driver.py`, `dummy_aggregate.py`, `dummy_slices.py`, `dummy_shell.py` and `dummy_xrn.py`. For these modules the functionality in C-Lab driver is the same than in Dummy driver. Apart from these modules, the C-Lab driver incorporates more modules as explained above. In the Dummy driver, the operations related to the SFA Registry were implemented in the `dummy_driver.py` itself. For the C-Lab driver, it was seen as a cleaner, more structured option to keep the SFA Registry operations in a separated module, following the style of the AM module. The other difference is the incorporation of the modules `clab_exceptions.py` and `clab_logging.py`. These are modules related to specific characteristics of the C-Lab testbed. The first module was included to handle potential exception raised in the `clab_shell.py` operations that operate with the REST API of the testbed. It was also thought as a good idea to keep a log file for the operations performed on the testbed by the SFAWrapper. The `clab_logging.py` module implements such feature.

The rest of the modules follow the template of the Dummy driver, but being specifically adapted to the C-Lab testbed when that was needed.

The Figure 4 shows the high-level components that integrate the SFA wrapper for Community-Lab. The Figure 5 shows with more detail the internal architecture of the wrapper, its different modules and the interaction among them and it graphically summarizes the explanation presented in this section.

4.5 Design decisions

Once adopted SFAWrap as a solution for federating C-Lab testbed with other experimental testbeds and facilities, there are some design decisions to be made for SFAWrap itself. Such decisions are related to the versions of SFA and RSpec. It was left to each testbed partner in Fed4FIRE to decide which version of SFA and RSpec to use in the implementation of the wrapper.

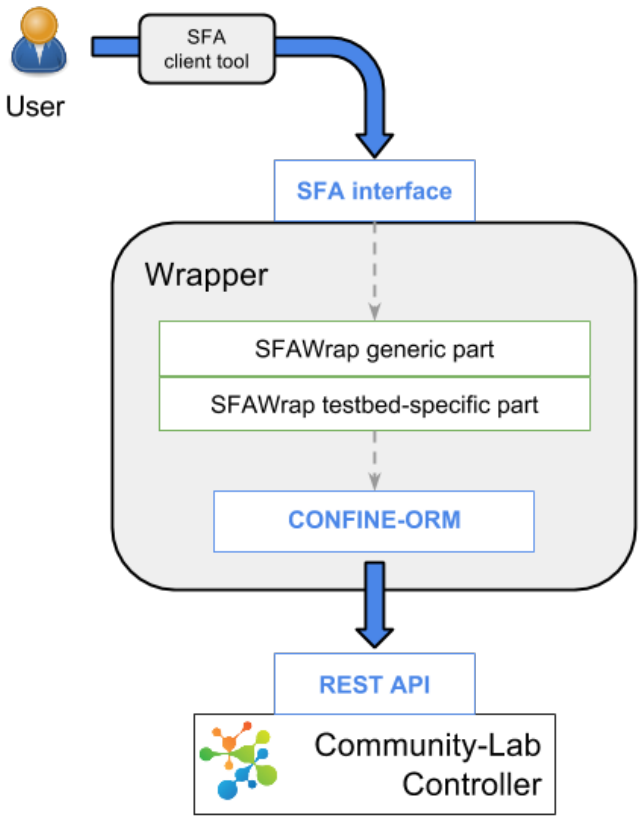

Figure 4: High-level components of Community-Lab SFA Wrapper and their interaction (Source: own creation)

4.5.1 SFA version

SFAWrap supports different versions of SFA: version 1, 2 and 3. These versions correspond to the API version of the Aggregate Manager (AM) component used when exposing the SFA interface through the SFAWrap. The AM API used in SFAWrap is defined by the GENI project. Different versions of the AM API are defined: AM API version 1 [59], AM API version 2 [60], AM API version 3 [58].

The version selected for C-Lab SFAWrap is GENI AM API version 3, so the AM interface exposed by the SFAWrap is compliant with this version. One of the reasons for the election is that the adoption of the newest current version was suggested from the European project. On the other hand, version 3 is the version that best adapts to C-Lab testbed and its model. One of the important changes between version 2 (and also version 1) and version 3 is that the operation of sliver creation is broken into 3 steps:

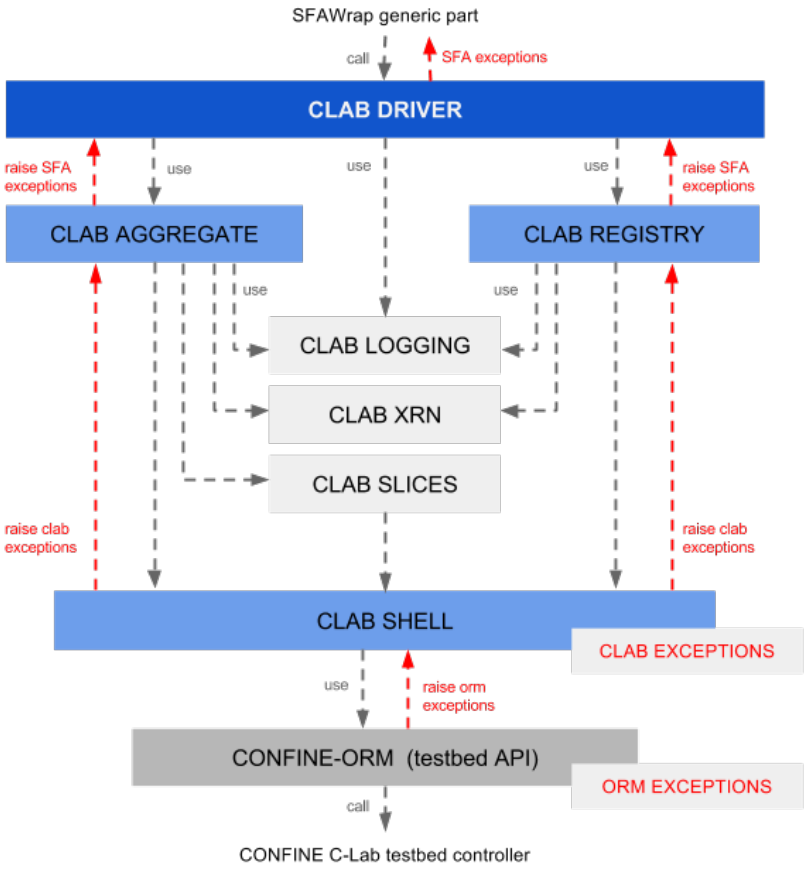

Figure 5: Architecture of the SFA Wrapper for Community-Lab (SFAWrap testbed-specific part) (Source: own creation)

1. **Allocate**. Reservation of resources
2. **Provision**. Instantiation of the resources
3. **PerformOperationalAction(geni_start)**. To start or boot the resources

This 3-steps process matches perfectly the process of sliver creation in C-Lab testbed: Register (Allocate), Deploly (Provision), Start (PerformOperationalAction(geni_start)). Therefore, AM API version 3 is seen as the best choice.

4.5.2 RSpec version

Resources are described in SFA and SFAWrap special XML documents called RSpec (resource specification). These RSpec documents follow XML schemas

that must be publicly available. SFAWrap uses by default standard schemas defined by the GENI project. There are different versions of these standard schemas: GENI RSpec version 2 [36], GENI RSpec version 3 [37].

Apart from the standard schemas, an Aggregate Manager can use its own schema to define its alternate RSpec format. Thus specific elements to describe testbed-specific characteristics can be added to the standard RSpec. The schema defining the new RSpec must be publicly available.

It a design decision to select the RSpec types and versions that the SFAWrap for C-Lab supports. It is necessary that the wrapper supports one standard RSpec type for compatibility reasons. The decision therefore, as was suggested from the project, was to support the newest RSpec standard version, RSpec version 3.

Extending the RSpec to define a specific schema for C-Lab in order to include some specific elements of the testbed was also considered as a possibility. This task however was left for later or future work since defining a new RSpec requires a considerable amount of work, no only for defining the new RSpec schema itself but also to adapt SFAWrap for using this new RSpec type.

5 Implementation

This chapter presents the implementation process of the federation tool, C-Lab SFA Wrapper, whose design was presented in the previous chapter. Section 5.1 introduces the environment used for such process and 5.2 the approach that drove this implementation. From section 5.3 to 5.11 (both included) the different modules that integrate the wrapper are explained. Section 5.12 explains how the wrapper is configured and which files are needed for this configuration. Finally, section 5.13 presents the most relevant challenges faced during the implementation and how they were solved.

5.1 Environment

SFAWrap for C-Lab was developed in an environment based on VCT (Virtual Confine Testbed). As explained in the background, VCT provides a simulation of the real C-Lab testbed in a local host, by using vitalization techniques (Linux Containers). It provides a suitable environment for developing the wrapper, since one can operate on the testbed and its controller without affecting the real testbed, and with the control that offers a localhost system.

Confine-ORM was another important element of the development environment and a key component to interact with the controller of the testbed through a programmatic interface. ORM was used not only as a library in the development of the code, but also for parallel testing via the Python console during the code development process.

The last important component of the development environment is the SFA package, which includes the SFAWrap code package. The implementation of the wrapper consists of extending the code of SFAWrap with a driver for C-Lab, with all the needed components. The command-line client tools of the SFA package were also an important element for parallel testing.

5.2 Approach

The development approach for the code of the wrapper combine a top-down analysis with a bottom-up implementation. The top-down approach is adopted in the first phase of the process to get an overview of the system. The result of this phase is a big picture of the wrapper, identifying the modules that will integrate it and their functionality. Such big picture is possible thanks to the previous analysis of existing wrappers for other testbeds that are used as a reference, and in particular, from the analysis of the Dummy flavour. The top-down approach was applied on the Dummy wrapper allowing the breaking down of the whole system into its modules and subsystems. The decomposition was carried out by following the flow of an SFA call on the wrapper through the different modules to understand their functionality. This first top-down approach of the process lead to the complete understanding of the system components and their behaviour.

The coding phase once the modules were identified was based on a bottom-up approach. The lower level modules, the closest to the testbed, were implemented as individual components. The modules were specified in detail and implemented (first implementation). The implementation was refined by a testing process throughout several rounds until the module worked correctly. Once properly implemented and tested, the different lower level modules were pieced together to build the more complex, higher level modules.

The motivation behind this combined approach is the following. On one hand, the fact of using a software package in which a generic part of the code is already implemented and that includes examples of the system being implemented, suggests the usage of a top-down approach. It is necessary to have an overview of the whole system and then shed light on the black boxes that integrate it, from higher level to lower level. Starting this analysis from the top allows to see which are the requirements for the lower level components; what these low level components need to provide to higher level ones. Thus, the analysis stops once it reaches low levels that directly depend on a particular testbed and its implementation, which may highly differ from the implementation of C-Lab testbed.

On the other hand, for the coding process a bottom-up approach is followed, implementing and testing first lowest level modules. The early testing process helps identifying errors soon and avoiding carrying errors around during integration of components. The wrapper and its components shows a hierarchical structure in which lower level components are used by higher level components, until reaching the generic level components. The requirements in terms of operations and functionality for low level components were obtained in the previous top-down phase. The requirements then can be mapped to a set of low level operations that the closest modules to the testbed must provide. These low level modules act as a library for higher level modules, thus building the complete stack of the system.

5.3 C-Lab Shell module

The C-Lab Shell module is a simple XML-RPC shell to the C-Lab testbed API. It provides high level methods wrapping the REST API of the testbed and can be seen as a high-level library to interact with the testbed.

The C-Lab Shell module uses the Confine-ORM high-level Python library. The idea behind this library and the shell module is basically the same: to offer a high-level programmatic API to interact with the testbed. The shell modules extends the functionality provided by the ORM library with more advanced and complex operations.

The implementation process consists of a first iteration in which the basic operations with general functionality were implemented. The implementation of such methods was simple, and mostly required the encapsulation of direct calls to the ORM library with some small modifications. This first iteration included the basic operations for the elements of the testbed: *get*, *update*, *create* and *delete* on nodes, slices, slivers, users, groups, templates (some operations only

are applicable for a subset of objects). The *get* methods offer filtering options to get a specific object or list of objects.

Subsequent iterations on the implementation process addressed operations with more specific functionality, related to some requirements or features that are useful for the wrapper. The need of implementing such operations arose in the process of developing other modules, such as the Aggregate Manager (`clab_aggregate.py`), the Registry (`clab_registry.py`) or the Slice Manager (`clab_slices.py`). The following list shows the most relevant operations and their functionality:

- Get the current state of a node or sliver.
- Get the IPV6 Management Network address of the sliver.
- Get available nodes for a slice (nodes in a correct state that do not contain a sliver for such slice).
- Renew methods for slices and slivers (update/extend their expiration date).
- Upload experiment data file to a slice or sliver. The experiment data file is used to upload an initialization script for the slivers that will be run at the end of the boot process.

5.4 C-Lab Exceptions module

The C-Lab Exceptions module defines the different exceptions that may arise from the usage of the Shell module. Some of the considered use cases for the shell module can lead to error situations. The exceptions defined in the module are raised by the methods of the shell method when the corresponding error occurs, so it can be handled by higher-level modules in the hierarchy.

Confine-ORM already catches some errors resulting from the interaction with the testbed and express them as Confine-ORM exceptions. Thus, some C-Lab exceptions consist of wrapping ORM exceptions whereas others deal specifically with errors not considered by Confine-ORM. The exceptions defined are:

- **IncorrectURI**, when retrieving any URI from the testbed that it is incorrect (malformed, unexisting or invalid).
- **OperationFailed**, when the operation being performed fails.
- **ResourceNotFound**, when the object requested is not found.
- **NotAvailableNodes**, when there are no available nodes for creating a sliver in a specific slice.

5.5 C-Lab XRN module

The C-Lab XRN module addresses the translation from C-Lab names to HRN or URN used in SFA. The module implements bidirectional translations for the names of all the objects in the testbed.

The implementation of this module is highly based on the Dummy XRN module. However there are some differences because of special cases that must be considered in the C-Lab scenario.

- **"@" in sliver identifier**. In C-Lab the identifier of the sliver has the format *slice_id*@*node_id*. The "@" is not valid for HRN and URN identifiers so when translating, it is replaced by the character "a".
- **"." in object names**. Some objects in C-Lab testbed are named using the character "." as a separator (specially user names). This causes problems when using HRN, since SFA interprets the "." as a separator for authorities and sub-authorities. Therefore, the "." is replaced by the sequence "__" when translating, thus creating a bidirectional translation.

5.6 C-Lab Aggregate module

C-Lab Aggregate module implements the Aggregate Manager (AM) interface corresponding to GENI AM API v3 [58] in which a list of methods and their functionality is specified. Each method was implemented using the previous implemented modules. The list of methods includes: `get_version`, `list_resources`, `describe`, `allocate`, `provision`, `perform_operational_action`, `renew`, `delete`, `shutdown` and `status`. The methods use calls to the Shell module to interact with the testbed, and using the XRN module to bidirectionally translate between C-Lab and SFA domains when needed.

Mapping the Sliver model

The AM offers methods to handle the resources of the testbed. As mentioned on previous section of the document, for the GENI AM API v3 the process of creating slivers is split in three steps. Each of these steps correspond to a specific method of the list: (1) `allocate`, (2) `provision`, (3) `perform_operational_action(geni_start)`. This process matches the process of creating slivers in C-Lab, also split in three equivalent steps: (1) `register`, (2) `deploy`, (3) `start`.

The call to each of the methods causes a change in the status of the resources, i.e. the slivers. This call is asynchronous (non-blocking) in the sense that it immediately returns without waiting for the status transition. Eventually, the status changes according to the performed operation. In the C-Lab testbed this behaviour is expressed as two parameters representing the status of a sliver: *set state* and *current state*.

The *current state* is the real state of the sliver whereas the *set state* is the desired state of the sliver. Thus, when a method is called to modify the state of a

sliver, the *set state* is changed and the methods returns immediately. Eventually this state will be applied to the sliver, so its *current state* will change, matching the *set state*.

The GENI AM API v3 defines a set of states in which a sliver can be throughout its creation process, the *Sliver Allocation states* [61]. Similarly, once the sliver creation process (allocation and provision) has finished the slivers are ready to be used. The GENI AM API v3 defines another set of states to describe the life-cycle of a sliver, the *Sliver Operational states* [62]. The transition between operational states is caused by *Sliver Operational actions* [63].

The GENI Sliver model (Creation process, Allocation states, Operational states, Operational actions) has its equivalent in the C-Lab domain. Apart from the 3-steps sliver creation process, C-Lab model also defines different actions that can be performed on the slivers and trigger changes on their state [64]. In the process of implementing the C-Lab Aggregate module, a complete mapping between the concepts of C-Lab and GENI Sliver model was done. Some concepts have their direct equivalent and some others are more complex. Table 2 shows the complete mapping between the two domains.

Mapping the RSpec

Another important task in the implementation of C-Lab Aggregate is to map the C-Lab concepts into a RSpec for describing resources of the testbed, i.e. nodes and slivers. All the methods in Aggregate module use directly or indirectly RSpecs. In C-Lab testbed API nodes and slivers have a set of parameters that describe their characteristics. These parameters must be expressed using a RSpec in the wrapper. As explained in Designed section the format of RSpec selected for the wrapper is GENI RSpec v3. Therefore, the parameters of nodes and slivers in C-Lab must be mapped into elements of the standard RSpec.

Such mapping includes the translation of the C-Lab object names to URN and HRN identifiers. Table 3 shows the details of this translation. The mapping of the C-Lab node parameters into the RSpec is showed in Table 4 and Table 5 shows the mapping for the C-Lab sliver.

Cache system

The Aggregate Manager module also implements a Cache system that caches the result of some read-only, very costly (time consuming) operations. In particular, the cache system is used with the `list resources` operation. This operation lists all the resources of the testbed that are managed by this AM, that is, the nodes of the testbed and some of their features. At this end, the AM interacts with the testbed controller to request the list of the nodes; then the AM adapts the node description to the SFA, RSpec model. This operation as described is very costly in terms of time. There are approximately 150 nodes in the testbed and the time for the controller to reply with the information of the nodes might be high, which leads to even higher time for the `list resources` operation (up to 180 sec).

C-Lab	GENI v3
Creation process actions	
Register	Allocate
Deploy	Provision
Start	PerformOpAction(geni_start)
Allocation states	
sliver does not exist yet	geni_unallocated
registered	geni_allocated
deployed	geni_provisioned
fail_alloc	geni_failed
fail_deploy	geni_failed
Operational states	
allocating	geni_pending_allocation
deploying	geni_notready
starting	geni_configuring
started	geni_ready
fail_start	geni_failed

Table 2: Mapping between C-Lab and GENI v3 Sliver models

Details of C-Lab names to XRN mapping	
Authority	clab
Sub-authority	clab
Chain of authorities	clab
HRN format	clab.object_name
URN format	urn:publicid:IDN+clab+obj_type+obj_name

Table 3: Details for the mapping from C-Lab names to URN and HRN identifiers

GENI RSpec v3	C-Lab Node
component_manager_id	urn of the AM authority
component_id	urn of the node
component_name	Name of the node
authority_id	urn of the Slice authority
sliver_id	urn of the sliver
available	true if current_state of the node is PRODUCTION
boot_state	translation of current_state of the node
hardware_types	architecture parameter of the node
interfaces	information of ifaces parameter of the node
slivers	RSpec for slivers of the node
services	login information containing information for ssh access to the sliver (username, ipv6 addr)

Table 4: Mapping between C-Lab node and GENI RSpec v3 Node

GENI RSpec v3	C-Lab Sliver
sliver_id	urn of the sliver
name	Name of the sliver
type	Constant value RD_sliver, which stands for Research Device sliver
disk_image	Information about the template of the sliver

Table 5: Mapping between C-Lab sliver and GENI RSpec v3 Sliver

The cache system improves this response time and reduces the load of the testbed controller. The result of `list resources` operation is cached for X minutes, being X a configurable parameter of the wrapper. By default, this time is 30 minutes. There is a trade-off between how often the wrapper queries the testbed controller (it discovers new, recently added nodes and discards removed nodes) and how up-to-date the results shown are. The value of 30 min is a reasonable value according to the stability of the tesbted and the necessary time to deploy new nodes.

5.7 C-Lab Registry module

The C-Lab Registry module implements the interface and the methods related to the management of the Registry. The Registry is a database that keeps track of the objects in the testbed. The SFAWrap generic code exports a Registry interface based on the Planetlab Slice Registry interface. The purpose of the C-Lab Registry module is to wrap the C-Lab specific Registry instance by exposing a set of methods that the generic Registry of SFAWrap will use. This set of methods implement the basic operations on the records of the registry: `register`, `update`, `remove` and also `update_relation` and `augment_records_with_testbed_info`. The implementation of these methods is carried out by using the Shell module that includes methods to interact with the Registry instance of the C-Lab testbed. The management of the records stored in the Registry also implies some translation between XRN identifiers and C-Lab specific names, which requires also the use of the C-Lab XRN module.

5.8 C-Lab Importer module

The C-Lab Importer module implements a very important and useful tool closely related to the C-Lab Registry module. It allows to import all the objects of the C-Lab testbed to the Planetlab Slice Registry of the SFAWrap so that both registries get synchronized. The import process consists of reading all the objects of the C-Lab Registry instance, translate all the records to SFA records and store them in the Planetlab Slice Registry of the SFAWrap.

The importer module is called automatically when starting the wrapper in order to perform an initial synchronization of the registries. From this point on, the registries are kept synchronized by the wrapper itself. The registry in SFAWrap is modified whenever an operation requires so; the C-Lab specific registry is updated automatically by the testbed server when the wrapper calls the specific operation on the testbed.

5.9 C-Lab Slices module

The C-Lab Slices module implements several methods to check the objects involved in the process of sliver creation. In particular, it offers methods to verify that the nodes and slices involved in the creation of a new sliver exist and are

in a correct state. The implementation of the methods in this module uses the Shell module to interact with the testbed Registry to perform the necessary verifications.

5.10 C-Lab Driver module

The C-Lab Driver module is the testbed-dependent, highest-level component of C-Lab SFAWrap. It provides the interfaces of the Aggregate and Registry by exposing their methods. The Driver module is the called by the generic modules of the wrapper. Its implementation consists of wrapping the methods of Aggregate and Registry modules.

5.11 C-Lab Generic module

The goal of C-Lab Generic module is to become a link between the generic part of the code and the C-Lab modules. The C-Lab Generic module defines which C-Lab modules implement the AM, Registry, driver, etc. To discover this relation generic modules use C-Lab Generic module. Thus, the implementation of such module basically consists of defining generic functions for each component, that return a reference to the specific module that implements it.

5.12 Configuration files

Once implemented the different modules that integrate the wrapper for the C-Lab testbed, there are some configuration steps to be done. Such steps consist of modifying the SFAWrap generic modules to add the ability of working with the new testbed wrapper, that is to configure the new testbed flavour. Minor modifications are needed in some files for this purpose.

Also a default configuration for the C-Lab flavour is added. The configuration includes some parameters such as the URL to access the testbed, the user and password and other specific parameters.

5.13 Challenges

Throughout the implementation process many challenges were identified. The next list presents the most relevant ones and how they were solved:

- **Current state of nodes and slivers.** The node description used in the REST API of the controller does not provide the *current state* of the nodes or slivers. In other words, the testbed controller does not know the *current state* of the nodes and their slivers. However, this parameter is required for the wrapper in some operations so the Shell implements two operations to get it. In these operations the management network address of the nodes is used to access them (or specific slivers that they contain) and send a GET request of the current state.

- **Automatic registration of new slices when allocating slivers.** In the federation scenario considered for the project, a user from another trusted administrative domain can use the resources of our testbed. Therefore, a user can allocate a sliver using a slice credential issued by other domain. This process implies the creation of an equivalent slice in C-Lab to map the slice of the trusted domain for which the sliver is being created. This slice is automatically registered in C-Lab testbed in the Allocate operation and the Registry instance of the wrapper is properly updated. The name given to the slice is the complete URN of the slice in the federated authority that is a unique identifier, so that name clash for slices is avoided in C-Lab.

- **Specification of User SSH keys in Provision.** In the Provision operation a user specifies the key(s) that will be later used for accessing the sliver via SSH. In C-Lab terms, it means that in the Deploy operation these keys need to be uploaded to the sliver automatically. At this end, the *experiment data* feature of C-Lab is used, which basically consists of uploading a compressed file with a directory tree that will be created in the sliver and as initialization script that will be run at bootstrap time. In the Provision operation of the wrapper an experiment data file with a customized script is created for each sliver. The script contains the public keys specified by the user, which are copied to the proper directory of authorized keys.

- **Provision reply with sliver login information.** According to SFA specification, the information to access an sliver is given to the user in the reply of the Provision operation, e.g. information for SSH access to the sliver. In the context of C-Lab it means that the IPv6 address of the sliver in the management network overlay (used for ssh-access to slivers) must be known at Deploy time, which it is not possible since the networking setup of slivers takes some time after triggering the Deploy operation. Fortunately the IPv6 address of the sliver follows a specific format and can be calculated using information that it is already known at Deploy time [65]. Thus, the IPv6 address of the sliver is calculated and included together with other parameters in the field `login` of Provision reply.

6 An Evaluation

This chapter presents an evaluation for the implemented C-Lab SFA Wrapper. The evaluation of our wrapper focuses on its behaviour, i.e. the validity of its behaviour, rather that its performance. The evaluation process presented in this chapter aims at validating the behaviour of the wrapper is the different scenarios and use case that it is going to work with. The sections 6.1, 6.2 and 6.3 present the three part of this evaluation process, which include a testing phase with generic SFA client tools, testing with the jFed tool and the execution of our own reference experiment.

6.1 Testing with SFA client tools

The SFA package includes two command-line tools to interact with the SFAWrap: an administration tool `sfaadmin.py`, and a client tool `sfi.py`. These tools were used for a first evaluation of the wrapper, testing its components and methods manually. Both tools are documented here [66].

sfaadmin.py

It is a command-line tool that allows the interaction with the wrapper within an administration context, that is, without using user credentials. `sfaadmin.py` utility skips the operation related to the credentials that are performed by generic components of the wrapper and it directly interacts with the components that manage the testbed.

This utility offers a set of basic operations on the Aggregate Manager component and the Registry component. The correct behaviour of the operations was manually checked by using the tool with the C-Lab SFAWrap.

sfi.py

It is a command-line client tool for SFAWrap that allows the interaction with the Aggregate Manager and Registry components, exposing the complete set of operations. Operations that act on the resources of the testbed or modify the state of the Registry require a user credential to guarantee permissions; other simpler query operations do not require credentials.

`sfi.py` utility acts as a XML RPC client that sends SFA calls with the necessary arguments to the XML RPC servers of the SFAWrap. After a generic processing of the calls in the SFAWrap (that includes the credential validation) the testbed-specific modules handle the call to apply the specific operation on the underlying testbed.

This utility provides complete AM API and also operations to manipulate the Registry. Credentials used with this tool were issued by the SFAWrap itself, thus belonging to the same administrative domain of the wrapper (no federation).

6.2 Testing with jFed

jFed [67] is a Java based framework developed by iMinds to support SFA testbed federation client tools. There are three different client applications based on the jFed framework: jFed Probe, jFed Automated Testing and jFed Experimenter. Probe and Automated-testing applications were used for the evaluation of C-Lab SFAWrap.

Configure jFed for C-Lab SFAWrap

To use jFed applications with C-Lab SFAWrap there are some previous steps that are necessary. jFed works with Emulab authority. iMinds provides an Emulab-based authority, Virtual Wall 2. It is necessary to get an account and download the certificate signed by Virtual Wall to work with jFed.

At the testbed side, the Virtual Wall 2 authority needs to be added as a trusted authority, so the wrapper accepts certificates signed by this authority. Moreover, the new authority for the C-Lab SFAWrap needs to be added to the list of configured authorities of jFed. This can easily be done by giving the URL of the AM and letting jFed scanning it to discover the configuration.

jFed Probe

jFed Probe is a client application to interact with SFA-compliant testbeds (or testbeds that expose somehow a SFA-compliant interface). The application offers a command-line version and a graphical user interface.

Similarly to the tests with `sfi.py`, the Probe application was used to manually test all the operations offered by the wrapper and validate their behaviour. The difference here is that the certificate and the credentials used in the operations are not issued by the Registry component of C-Lab SFAWrap, but by another authority that is accepted as a trusted authority. Therefore, the Probe application allows to test the wrapper in a federation scenario.

jFed Automated Testing

jFed Automated Testing is a tool to run compliance tests for the different SFA components of the testbeds, with a command-line and graphical user interface. It offers a complete suite of tests for the Aggregate Manager and Slice Authority, testing the different methods of the components and reporting results with information about the failures. This is a very useful tool for the validation task of the wrapper.

iMinds Automated Scenario Tests

iMinds offers a website with automated scenario tests for testbeds and wrappers that are publicly available. These scenarios consist of a set of tests and experiments to validate the behaviour of the testbeds, similarly to the Automated testing application, which automatically, daily run.

For C-Lab SFAWrap to be included in this automatic testing scenario there is a required previous step for the machine that runs the test. The machine needs to join the COFNINE Management Network overlay (that is, install and configure a tinc client) to be able to access nodes and slivers through their management address. Once this step was done, C-Lab was added as a new default authority in the jFed release and added also to the automated testing scenarios.

6.3 Reference experiment

A Reference Experiment was also run as a part of the evaluation process. The reference experiment aims to validate the behaviour of the wrapper in a federation scenario by allocating certain resources of the testbed and then test their functionality to guarantee that the wrapper worked as expected. At this end, the reference experiment was designed with three steps:

1. Resource allocation. Using a credential of a federated authority (Virtual Wall 2), a new slice is registered in C-Lab SFAWrap and two new slivers are instantiated.

2. Dummy experiment. Once the slivers are ready for running experiments, a very simple experiment is perform on them to guarantee their correct behavior. The experiment consists of one sliver sending three ping messages to the other sliver, by running the Linux command `ping6` remotely.

3. Report results. Finally, the results of the experiment are reported by creating a folder with two text files: a file containing the details of the resource allocation process as well as details about the created slivers; a file containing the results of the ping command.

The reference experiment is completely automatic, which means that it runs the 3 steps mentioned above without any user interaction. Before starting the experiment there are some configuration parameters that must be set (or left as default). The reference experiment is implemented in Java, leveraging different libraries that provide useful functionality for the experiment.

For the interaction with the wrapper to create the new slice and slivers, the source code of the jFed Library is used. This library provides methods to establish connections to authorities from the jFed Authority List, get credentials signed by these authorities and allocate resources on the testbeds that they represent.

For running the experiment in the slivers once they are ready to use, the JSch from JCraft is used. This library provides a pure Java implementation of SSH2 and allows to connect to an ssh server, transfer files and run remote commands among other features. The simple `ping` experiment is implemented as a remote `ping6` command from one sliver to the other. The IPv6 addresses of the slivers (corresponding to the IPv6 Management Network overlay implemented by tinc) are an outcome from the first allocation step of the reference experiment.

The report of the results basically consists of generating a new directory with the same name as the slice and generate two text files inside. The file `sliverSetupDetails.txt` contains the results of the allocation process, including details about the duration of each step as well as details about urns and addresses of the created slivers. The file `pingDetails.txt` contains the result of remotely running the command "`ping6 -c3 ipv6_addr_of_sliver2`"" on the sliver 1.

The results of the Reference Experiment were successful; the allocation process finished correctly and the slivers created were demonstrated to work properly by running the simple ping experiment successfully.

7 Conclusions

This thesis addresses the federation of Community Networking testbeds. The concept of federation and its implications in this scenario has been analysed. Federation aims at providing interoperability among testbeds and it can address different abstraction levels. Such interoperability requires homogeneity or standardization on the facilities, which has to be compatible with the leveraging of their heterogeneous features.

In particular this thesis has focused on the federation of the Community-Lab testbed with other testbeds and facilities. At this end and according to the standardization mentioned above, a new software layer has been developed on top of the testbed to provide an interface based on the SFA federation standard for testbeds. Thus, the main contribution is the development of a C-Lab SFA wrapper software to enable federation of Community-Lab with other testbeds through SFA.

The remaining of this chapter is organized as follows: a more detailed list of contributions is presented in section 7.1. A discussion about the results achieved is presented in section 7.2. Finally, sections 7.3 and 7.4 address the lessons learned and the future work respectively.

7.1 Contributions

This section summarizes the contributions made as part of this thesis.

C-Lab SFA wrapper

A SFA interface for Community-Lab testbed was developed. Such interface is a wrapper based on the SFAWrap code, which basically provides a SFA AM interface. The wrapper was deployed on a server for public access.

C-Lab SFA wrapper AM	`https://84.88.85.16:12346`

Automated testing results

The project that serves as a context for the development of the SFA wrapper for Community-Lab offers a website to show the results of automated tests run on the deployed wrappers to validate its behaviour and functionality. Since the C-Lab SFA wrapper was deployed on a server, it was included to the set of wrappers for automated testing.

Results of automated testing for the C-Lab SFA wrapper	`https://flsmonitor.fed4fire.eu/`
Results of Login automated test for the C-Lab SFA wrapper	`http://monitor.ilabt.iminds.be/scenarios.php?filter=fed4fire`

Code of the wrapper

The code implemented throughout the thesis is publicly available in Github repositories. The code implemented includes source code for the wrapper written in Python and the reference experiment for the wrapper written in Java.

Source code of C-Lab SFA wrapper	`https://github.com/gnogueras/sfa`
Source code of the Reference Experiment	`https://github.com/gnogueras/refexp`

Documentation in the CONFINE Wiki

Documentation about the usage and the architecture of the C-Lab SFA wrapper was included to the Wiki of the testbed. A tutorial about using the jFed tool with the wrapper was also written for the Wiki.

Usage of the wrapper	`https://wiki.confine-project.eu/usage:sfawrapper`
Architecture of the wrapper	`https://wiki.confine-project.eu/usage:sfawrapper`
Tutorial jFed and wrapper	`https://wiki.confine-project.eu/tutorials:sfawrapper`

Documentation in the Fed4FIRE website

A short explanation introducing the Community-Lab testbed was written for the Fed4FIRE testbeds documentation website.

Explanation about C-Lab in the Fed4FIRE website	`http://doc.fed4fire.eu/testbeds.html#community-lab-c-lab`

7.2 Discussion

7.2.1 The wrapper

The SFA wrapper for Community-Lab works correctly, as the different tests of the evaluation process have shown. The wrapper provides the user with another interface (instead of the Dashboard) to interact with the testbed. The interface exposes the set of operations defined by SFA. These operations allow the user to use and manage the resources offered by the testbed. All the operations work properly (as expected in SFA) except the `Renew` operation that has certain limitations. Due to restrictions from the Community-Lab testbed itself, the expiration date of slices and slivers can be set to a maximum of 30 days from the current date. Therefore, attempts to set the expiration date beyond this limit (that could be requested through the `Renew` operation of SFA) will not have any effect.

The main limitation of the implemented wrapper is the reduction of the customization capability when instantiating new slivers. In the Community-Lab testbed there are many customizable aspects of the sliver such as network interfaces, template, experiment data, overlay, etc. Most of these features are not customizable through the SFA wrapper, which means that the user cannot choose or specify them when interacting with the testbed through the wrapper. This is because the wrapper only supports the standard GENI RSpec v3 to describe the resources, i.e. slivers, that the user is instantiating. The customizable features of the slivers do not have any equivalent element or field in the standard RSpec, so they cannot be mapped to SFA domain. Currently, the wrapper solves this problem using a default configuration for the sliver parameters that are not specified in the RSpec. The design of a new RSpec for Community-Lab (perhaps an extension of the standard GENI RSpec v3) would solve this problem. Such task is left as future work.

Another limitation is related to the management of the slices through the wrapper. The wrapper for Community-Lab provides a SFA AM that allows to manage the resources of the testbed, i.e. nodes and slivers. Through the AM of the wrapper a user can register, deploy, start, stop, update, renew and delete slivers in the testbed. The AM, by definition, does not manage slices, so the wrapper does not provide any SFA interface to manage slices. However in the Community-Lab testbed, a new sliver can only be created in the context of a slice. In other words, a user can create a new sliver only for an existing slice. In a federation scenario such slice will belong to a federated authority and will contain slivers of multiple federated testbeds. The creation of a sliver through the SFA wrapper requires to present the credential of the slice issued by the federated, trusted authority. The wrapper creates then an equivalent (proxy) slice in the Community-Lab testbed, so the new sliver can be allocated in Community-Lab within the context of such equivalent slice. The problem of this behaviour is that the SFA user has no way of deleting the equivalent slice through the SFA wrapper. Even after deleting all the slivers in the slice, the slice will remain in the Community-Lab testbed until it expires and is automatically

deleted. The problem does not seem to be critical and it can be tolerated, because slices do not use any resources.

7.2.2 SFAWrap

The development of a SFA wrapper based on the SFAWrap software package was a suggestion from the project to provide a SFA interface for testbed federation. Other solutions (particular solutions not based on SFAWrap) could also be considered, as some testbed partners of the project used them. In the case of Community-Lab the usage of SFAWrap helped and facilitated the work a lot. SFAWrap includes modules dealing with the generic functionality implied on the wrapper, leaving only the testbed-specific modules to be implemented by the developer. However, its usage also entailed some limitations and problems.

First limitation is the lack of documentation of the written code, which obstructed and delayed its understanding. The needed testbed-specific modules and their functionality and requirements, as well as the functionality of the generic, existing modules was hard to analyze and understand. Good documentation about the existing code would have help a lot for such task.

Another limitation of SFAWrap is related to the way that generic modules handle the credentials. The credential validation performed in any operation is done by the generic part of the code. Once the credential is properly validated, the generic modules invoke the testbed-specific modules that act on the testbed. The problem is that no information about a credential or the user performing the action is passed to the testbed-specific modules. This leads to an anonymous use of the testbed by the wrapper. It would be desirable to receive the information about which user invokes each operation on the testbed using the wrapper. Such requirement was notified to the developers of SFAWrap that considered as a positive feedback although the problem is not yet addressed.

Finally, although it is not supposed to be modified, some small modifications were done in the generic part of the SFAWrap code to fulfill all the requirements from the testbed federation project.

7.2.3 Testbed federation

Testbeds have become an important facility for the Future Internet Research and Experimentation, since they are a good stage between simulation and production covering all the areas of the future Internet. If federation is added to such scenario, then the range of possibilities gets much wider. Testbed federation allows to build a global aggregate of heterogeneous facilities, interacting and leveraging each others' capabilities. Thus, more complex experiments can be done on a realistic infrastructure built from a combination of testbeds.

Federation provides many new possibilities and benefits, but also a lot of challenges. Federation does no only mean to build a common interface to interact with multiple facilities, but enable to work with them as if they were a single facility. Therefore, this federation might be very complex when addressing aspects such as accounting, permissions, networking and mapping of specific

features. Standardization in terms of architectures, APIs and interfaces helps and is needed, despite the difficulty that defining standards in a so potentially heterogeneous scenario represents. The heterogeneity of the facilities is what gives the richness in terms of offered possibilities and capabilities. The limitations of the federation will be related to how well the federated system is able to exploit such richness while achieving the desired level of homogeneity.

7.3 Lessons learned

The following list summarizes the conclusions and the lessons learned throughout the development of the thesis:

- Federation is usually understood as federation of existing facilities. That was the case of Community-Lab, an existing testbed in production state. For Community-Lab, federation was taken into account from the beginning and its architecture and model were designed to facilitate a potential federation in the future. It is good to have potential future possibilities of a facility from the initial design phase.
- Federation implies a trade-off between the level of homogeneity achieved and the ability of leveraging the heterogeneous capabilities of the federated facilities
- Reuse existing code might help a lot. Therefore, to document and comment the code properly is a very important task; it will help other people that want to reuse it.
- The evaluation and validation is a very important part of the code development. Tests for all the use cases are needed, however your evaluation will be "an evaluation", as there are may hard to capture small details.

7.4 Future work

During the implementation, the evaluation and the analysis of its limitations, some improvements on the Community-Lab SFA wrapper were identified. They have been left as future work:

- New RSpec for C-Lab. In order for the wrapper to be able to exploit all the capabilities that the Community-Lab testbed provides, a new RSpec could be defined in which all the features and customizable aspects are properly mapped. Such RSpec might be an extension of the standard GENI RSpec v3, including new elements and fields to offer the user the possibility of customizing their slivers.
- Include modifications on the generic code of SFAWrap to enable the pass of credentials to the testbed-specific part of the code. This way it would be possible to identify the user invoking each operation and avoid a completely anonyomous usage of the wrapper.

- Consider the possibility of using the SFA Registry interface (PlanetLab-SliceRegistry instance) that the SFAWrap implements as a Slice Authority, and add this authority to the federated authorities of the project. That would mean to have a Community-Lab Slice Authority that can issue and sign credentials for users and slices; these credentials can then be used in the trusted, federated testbeds.

Appendix: User Manual

Compatible OS

The compatibility of the wrapper for C-Lab with different operating systems is bound to the compatibility of their components, specially the SFA package. Thus the supported OS for the wrapper are:

- Debian OS: Linux Ubuntu (from version 12.04)
- Linux Ubuntu based OS, for example, Linux Mint
- Fedora (from version 14)

A requirement for the OS is to have installed Python 2.7, since ORM library and the wrapper itself are implemented using such version.

In this thesis the wrapper have been developed and deployed in a Linux Ubuntu Desktop 13.04 (64 bits), for compatibility reasons with SFA at the time the thesis started. Nowadays, newer versions of Ubuntu are correctly supported.

Installation of C-Lab SFAWrap

This section explains in detail the installation process of the federation tool C-Lab SFAWrap in Linux Ubuntu Desktop 13.04. The installation process has some dependencies (python-pip, git) that need to be installed to successfully complete. Such dependencies can be installed using the following commands:

```
apt-get install postgresql python-all-dev
apt-get install python-pip git
```

Installation process of C-Lab SFAWrap:

1. Install the dependencies for C-Lab SFAWrap
   (a) Install SFA packages as explained in [66]

```
echo "deb http://build-debian.onelab.eu/sfa/stable-
    sfa3-raring-64/ ./" $>$ /etc/apt/sources.list.d/
    sfa.list
apt-get update
apt-get install --force-yes sfa sfa-common
apt-get install --force-yes sfa-client sfa-dummy
```

   (b) Install CONFINE-ORM library as explained in [68]

```
  pip install confine-orm
```

2. Clone C-Lab SFAWrap repository from GitHub:
3. Patch SFAWrap with C-Lab code

(a) Copy `clab/` directory to `/usr/lib/python2.7/dist-packages/sfa/`

(b) Copy `clab_importer.py` to `/usr/lib/python2.7/dist-packages/sfa/importer/`

(c) Copy `clab.py` to `/usr/lib/python2.7/dist-packages/sfa/generic/`

4. Copy default configuration file `default_config.xml` to `/etc/sfa/`

5. Modify the files: `sfa-config-tty` (and `sfa-config`, `configs/site_config`)

Configuration of C-Lab SFAWrap

Once the installation process is finished, the SFAWrap tool needs to be configured to work with C-Lab testbed. In this section the configuration process is explained in detail.

Configuration process of SFAWrap to work with C-Lab:

1. Run the SFA configuration tool:

```
sfa-config-tty
```

2. Select the option for modifying the configuration:

```
u    # for usual changes
```

3. Enter the following configuration parameters:

```
sfa_generic_flavour : [clab]
sfa_interface_hrn : [clab]
sfa_registry_root_auth : [clab]
sfa_registry_host : [84.88.85.16]
sfa_aggregate_host : [84.88.85.16]
sfa_sm_host : [84.88.85.16]
sfa_db_host : [84.88.85.16]
sfa_clab_user : [sfawrap]
sfa_clab_password : [********]
sfa_clab_group : [fed4fire]
sfa_clab_url : [https://controller.community-lab.net/
   api/]
sfa_clab_auto_slice_creation : [True]
sfa_clab_auto_node_creation : [False]
sfa_clab_aggregate_caching : [True]
sfa_clab_aggregate_cache_expiration_time : [1800]
sfa_clab_default_template : [Debian Squeeze]
sfa_clab_temp_dir_exp_data : [/home/gerard/clab_sfawrap
   /experiment-data/]
```

4. Make the configuration persisting:

```
w    # to write the changes
```

5. Restart the SFA wrapper to apply the configuration:

```
r     # to restart
```

6. Exit:

```
q     # to quit
```

Apart from the configuration of the wrapper there is another necessary step. The host where the wrapper is installed needs to have access to the nodes and slivers, for example ssh access. In particular, the wrapper needs to access to the IPv6 overlay that serves as a management network [69] for the testbed and connects nodes, slivers and servers. Such overlay is setup using the software *tinc* [70] as a backend [71]. A new host can be configured to access this overlay network and receive a unique IPv6 address. Thus the host can access any node or sliver through this overlay network. The following lines summarize the steps to setup the tinc client and the IPv6 overlay (complete information in the Manual [72]):

7. Get an account in Community-Lab and access the Dashboard [27]
8. Add the new host in the "TINC HOSTS" section.
9. Get customized instructions via the "Help" button in the page of the new added host. Follow the instructions to complete the process.

jFed with C-Lab SFAWrap

Installation of jFed

jFed is SFA client-application in Java that has been used as a reference application to interact and test the wrapper throughout the thesis. jFed is developed by iMinds and complete information about its installation can be found here [67].

NOTE: jFed was under continuous development during the realisation of the thesis. The jFed release used for the thesis was r1257.

Configuration of jFed with C-Lab SFAWrap

jFed includes a default list of configured authorities to work with. The C-Lab SFAWrap AM will be included to the default list in future releases.

However, a user can manually edit the local authority list and add new authorities. jFed provides a scanner tool that makes this task very easy. The scanner is given the URL of the AM that is willing to be added, and it automatically detects its configuration.

URL of the C-Lab SFAWrap AM to scan: `https://84.88.85.16:12346`

Otherwise the configuration for the new authority can be set manually:

Name: `Community-Lab Controller Wrapper`
URN: `urn:publicid:IDN+clab+authority+am`
URN part: `clab`
Allowed Certificate: content of the file `clab.gid`
Allowed Certificate Aliases: `clab`
Authority Server URL's:

Role	version	URL
AM	3	`https://84.88.85.16:12346`
PlanetLabSliceRegistry	1	`https://84.88.85.16:12345`

Probe GUI Tool

jFed Probe GUI tool is the client application to interact with the configured authorities. A user needs to present a certificate issued by an authority to Login. This certificate will be used to get the credentials from the corresponding Slice Authority. As explained earlier in the document, the C-Lab SFAWrap AM accepts credentials from the federated authority Virtual Wall. To get these credentials a user needs and account and certificate from Emulab. Complete information about the process here [73].

Example of usage

This example shows the process of how to create a new slice with a new sliver in Community-Lab testbed using jFed Probe GUI tool and the C-Lab SFAWrap AM. The example assumes that jFed is correctly configured to work with C-Lab SFAWrap AM and that the user has a valid certificate from the Emulab Authority. After starting jFed Probe GUI and Login with the certificate:

1. **Get userCredential from the iMinds Virtual Wall 2 Authority**. Select the operation `getCredential` from the *ProtoGeni SA* interface, in *User amd Slice APIs* menu. Call the operation on the authority iMinds Virtual Wall 2 of the default authority list. The operation will return the user credential; the credential can be seen in the *Processed Geni Reply Value* tab of the reply.

2. **Register a new slice in the iMinds Virtual Wall 2 authority**. Select the operation `register` from the *ProtoGeni SA* interface, in *User amd Slice APIs* menu. Specify as user credential the credential from step 1 and set the slice name. Call the operation on the authority iMinds Virtual Wall 2 of the default authority list. The operation will return the slice credential of the new registered slice; the credential can be seen in the *Processed Geni Reply Value* tab of the reply.

3. **Allocate a new sliver in the C-Lab AM using the slice credential from iMinds Virtual Wall 2 Authority**. Select the operation `allocate` from the *Aggregate Manager v3* interface, in *Aggregate Manager APIs* menu. Select the slice credential from step 2 in the credential list and set the slice URN properly. Specify a valid request RSpec to describe the desired slivers (examples of RSpecs can be found elsewhere []). Call the operation on the authority of C-Lab SFAWrap AM. The operation will return a Geni Reply Value with information about the allocated sliver (URN, state and expiration date), and also a manifest Rspec and the slice URN. This information can be seen in the *Geni Reply Value* tab of the reply.

4. **Provision the allocated sliver in the C-Lab AM using the slice credential from iMinds Virtual Wall 2 Authority**. Select the operation `provision` from the *Aggregate Manager v3* interface, in *Aggregate Manager APIs* menu. Select the URN of the allocated sliver from the list of URNs and the slice credential from step 2 in the credential list. In the *users* section select the user and add a new SSH public key that will be uploaded to the sliver for later SSH access. Call the operation on the authority of C-Lab SFAWrap AM. The operation will return a Geni Reply Value similar to the previous one, with updated information about the provisioned sliver. This information can be seen in the *Geni Reply Value* tab of the reply.

5. **Start the provisioned sliver in the C-Lab AM using the slice credential from iMinds Virtual Wall 2 Authority**. Select the operation `performOperationalAction` from the *Aggregate Manager v3* interface, in *Aggregate Manager APIs* menu. Select the URN of the provisioned sliver from the list of URNs and the slice credential from step 2 in the credential list. In the *action* section specify `geni_start` (it is the action by default). Call the operation on the authority of C-Lab SFAWrap AM. The operation will return a Geni Reply Value with updated information about the sliver, indicating the allocation and operaitonal status. This information can be seen in the *Geni Reply Value* tab of the reply.

6. **Wait for the sliver to be Ready**. After the `performOperationalAction` operation, the sliver will be eventually ready for being used. To check it the sliver is ready the operation `status` can be called. Select the operation `status` from the *Aggregate Manager v3* interface, in *Aggregate Manager APIs* menu. Select the URN of the created sliver from the list of URNs and the slice credential from step 2 in the credential list. Call the operation on the authority of C-Lab SFAWrap AM. The operation will return a Geni Reply Value with updated information about the sliver, indicating the allocation and operational status. If the `geni_operational_status` field has the value `geni_ready`, the sliver is ready for being used. This information can be seen in the *Geni Reply Value* tab of the reply.

7. **Obtain Login information of the sliver for SSH access**. Last step once the sliver is created and ready for usage is to access the sliver via SSH. The login information of the sliver, which is basically its IP address, can be seen in the Manifest RSpec of the `Provision` Geni Reply Value. Moreover, the operation `describe` also return this information. Select the operation `describe` from the *Aggregate Manager v3* interface, in *Aggregate Manager APIs* menu. Select the URN of the created sliver from the list of URNs and the slice credential from step 2 in the credential list. Make sure that the *compressed* option is unchecked (otherwise the RSpec in the reply will be compressed). Call the operation on the authority of C-Lab SFAWrap AM. The operation will return a Geni Reply Value with information about the sliver, including a Manifest RSpec in which the Login information for the sliver (its IPv6 address) is specified. This information can be seen in the *Geni Reply Value* tab of the reply.

8. **SSH access to the created sliver**. If the tinc client is correctly configured in your host as explained on previous sections, the IPv6 address of the sliver can be used to access it via ssh. In the ssh command, specify the private key corresponding to the public key that was uploaded to the sliver in the `provision` operation, step 4.

References

[1] B. Braem, R. Baig Viñas, A. L. Kaplan, A. Neumann, I. Vilata i Balaguer, B. Tatum, M. Matson, C. Blondia, C. Barz, H. Rogge, F. Freitag, L. Navarro, J. Bonicioli, S. Papathanasiou, and P. Escrich, "A case for research with and on community networks," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 3, p. 68, Jul. 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2500098.2500108

[2] "guifi.net." [Online]. Available: http://guifi.net/

[3] "AWMN ." [Online]. Available: https://www.awmn.net/content.php

[4] "FunkFeuer." [Online]. Available: http://www.funkfeuer.at/

[5] A. Neumann, I. Vilata, X. León, P. Escrich, L. Navarro, and E. López, "Community-lab: Architecture of a community networking testbed for the future internet," in *International Workshop on Community Networks and Bottom-up-Broadband (CNBuB 2012)*, IEEE Press. IEEE Press, 10/2012 2012.

[6] M. Serrano, S. Van der Meer, V. Holum, J. Murphy, and J. Strassner, "Federation, a matter of autonomic management in the future internet," in *Network Operations and Management Symposium (NOMS), 2010 IEEE*, April 2010, pp. 845–849.

[7] J. Putman, *Architecting with rm-odp*. Prentice Hall Professional, 2001.

[8] M. Bearman and K. Raymond, "Federating Traders: An ODP Adventure." *Open Distributed Processing*, 1991. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.33.9116&rep=rep1&type=pdf

[9] J.-P. Deschrevel, "The ansa model for trading and federation," *Architecture Report APM*, vol. 1005, 1993.

[10] J. Famaey and F. De Turck, "Federated management of the future internet: status and challenges," *International Journal of Network Management*, vol. 22, no. 6, pp. 508–528, 2012. [Online]. Available: http://dx.doi.org/10.1002/nem.1813

[11] S. Wahle, B. Harjoc, K. Campowsky, T. Magedanz, and A. Gavras, "Pan-European testbed and experimental facility federation – architecture refinement and implementation," *International Journal of Communication Networks and Distributed Systems*, vol. 5, no. 1/2, p. 67, Jul. 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1831352.1831356

[12] T. Kurze, M. Klems, and D. Bermbach, "Cloud federation," *CLOUD COMPUTING ...*, 2011. [Online]. Available: http://www.thinkmind.org/index.php?view=article&articleid=cloud_computing_2011_2_20_20114

[13] W. Vandenberghe, "Architecture for the heterogeneous federation of future internet experimentation facilities," *Future Network and . . .*, 2013. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6633558

[14] "Community Lab." [Online]. Available: http://www.community-lab.net/

[15] "CONFINE project." [Online]. Available: http://confine-project.eu/

[16] "CONFINE Wiki. Architecture." [Online]. Available: http://wiki.confine-project.eu/arch:overall

[17] Upc, universitat politècnica de catalunya. [Online]. Available: http://www.upc.edu

[18] Kth, royal institute of technology. [Online]. Available: http://www.kth.se

[19] University of rome tor vergata. [Online]. Available: http://web.uniroma2.it

[20] Pangea. [Online]. Available: http://www.pangea.org

[21] Sics, swedish institute of computer science. [Online]. Available: http://www.sics.se

[22] iminds. [Online]. Available: http://www.iminds.be

[23] "CONFINE Wiki. Testbed Node Architecture." [Online]. Available: http://wiki.confine-project.eu/arch:node

[24] "CONFINE Wiki. REST-API." [Online]. Available: http://wiki.confine-project.eu/arch:rest-api

[25] "Confine-orm documentation." [Online]. Available: http://confine-orm.readthedocs.org/en/latest/

[26] "CONFINE Wiki. VCT." [Online]. Available: http://wiki.confine-project.eu/soft:node-system-bare-bones#vct

[27] "Confine community-lab testbed management." [Online]. Available: https://controller.community-lab.net

[28] "Sandbox testbed management." [Online]. Available: https://sandbox.confine-project.eu

[29] "Sandbox (software development) from wikipedia." [Online]. Available: http://en.wikipedia.org/wiki/Sandbox_(software_development)

[30] "Fed4FIRE project." [Online]. Available: http://www.fed4fire.eu/

[31] L. Peterson, S. Sevinc, J. Lepreau, R. Ricci, J. Wroclawski, T. Faber, S. Schwab, and S. Baker, "Slice-Based Facility Architecture," PlanetLab, Tech. Rep., 2009. [Online]. Available: https://github.com/planetlab/sfa/blob/master/docs/sfa.pdf

[32] "CONFINE Wiki. SFA." [Online]. Available: http://wiki.confine-project.eu/sfa:start

[33] "CRUD Operations, Wikipedia ." [Online]. Available: http://en.wikipedia.org/wiki/Create,_read,_update_and_delete

[34] "SFA Wrap ." [Online]. Available: http://sfawrap.info/

[35] "Protogeni RSpec." [Online]. Available: http://www.protogeni.net/ProtoGeni/wiki/RSpec

[36] "GENI RSpec v2." [Online]. Available: http://www.protogeni.net/ProtoGeni/wiki/RSpecSchema2

[37] "GENI RSpec v3." [Online]. Available: http://www.geni.net/resources/rspec/3/

[38] "GENI SFA credential version 2." [Online]. Available: http://groups.geni.net/geni/wiki/GeniApiCredentials

[39] "GENI SFA credential version 3." [Online]. Available: http://groups.geni.net/geni/wiki/GeniApiCredentials

[40] "GENI SFA ABAC credential." [Online]. Available: http://groups.geni.net/geni/wiki/TIEDABACCredential

[41] "CorteXlab." [Online]. Available: http://www.cortexlab.fr/

[42] "FEDERICA." [Online]. Available: http://www.fp7-federica.eu/

[43] "IoTLab." [Online]. Available: http://www.iotlab.eu/about.php

[44] "NITOS Wireless Testbed." [Online]. Available: http://nitlab.inf.uth.gr/NITlab/index.php/testbed

[45] "OpenStack." [Online]. Available: https://www.openstack.org/

[46] "PlanetLab Project." [Online]. Available: https://www.planet-lab.org/

[47] "Microkernel from Wikipedia." [Online]. Available: http://en.wikipedia.org/wiki/Microkernel

[48] R. Bush and D. Meyer, "Rfc 3439: Some internet architectural guidelines and philosophy," 2003.

[49] P. Escrich and R. Baig, "Wibed, a platform for commodity wireless testbeds," *Wireless Days (WD), ...*, 2013. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6686492

[50] "CONFINE Wiki. Wibed." [Online]. Available: https://wiki.confine-project.eu/wibed:start, urldate = 10/05/14

[51] L. Navarro, A. Kaplan, J. Bonicioli, E. López, and I. Vilata, "Deliverable D2.6. Federation mechanisms for community networks," 2013. [Online]. Available: http://confine-project.eu/files/2013/09/D2.6.pdf

[52] "PlanetLab from Wikipedia, the free encyclopedia." [Online]. Available: http://en.wikipedia.org/wiki/PlanetLab

[53] "PlanetLab Central." [Online]. Available: https://www.planet-lab.org/

[54] "PlanetLab Europe." [Online]. Available: http://www.planet-lab.eu/

[55] "OneLab Project." [Online]. Available: http://www.onelab.eu/

[56] L. Peterson, S. Sevinc, S. Baker, T. Mack, R. Moran, and F. Ahmed, "PlanetLab Implementation of the Slice-Based Facility Architecture," PlanetLab, Tech. Rep., 2009. [Online]. Available: https://github.com/planetlab/sfa/blob/master/docs/sfa-impl.pdf

[57] "CONFINE Wiki. Federation." [Online]. Available: http://wiki.confine-project.eu/federation:start

[58] "GENI Aggregate Manager API Version 3." [Online]. Available: http://groups.geni.net/geni/wiki/GAPI_AM_API_V3

[59] "GENI Aggregate Manager API Version 1." [Online]. Available: http://groups.geni.net/geni/wiki/GAPI_AM_API_V1

[60] "GENI Aggregate Manager API Version 2." [Online]. Available: http://groups.geni.net/geni/wiki/GAPI_AM_API_V2

[61] "GENI AM API v3 Sliver Allocation States." [Online]. Available: http://groups.geni.net/geni/wiki/GAPI_AM_API_V3/CommonConcepts#SliverAllocationStates

[62] "GENI AM API v3 Sliver Operational States." [Online]. Available: http://groups.geni.net/geni/wiki/GAPI_AM_API_V3/CommonConcepts#SliverOperationalStates

[63] "GENI AM API v3 Sliver Operational Actions." [Online]. Available: http://groups.geni.net/geni/wiki/GAPI_AM_API_V3/CommonConcepts#SliverOperationalActions

[64] "Confine wiki. slice and sliver states." [Online]. Available: http://wiki.confine-project.eu/arch:slice-sliver-states

[65] "CONFINE Wiki. Addressing." [Online]. Available: http://wiki.confine-project.eu/arch:addressing

[66] "SFATutorial: Installation process." [Online]. Available: http://svn.planet-lab.org/wiki/SFATutorialInstall

[67] "jFed iMinds." [Online]. Available: http://jfed.iminds.be/

[68] "CONFINE-ORM Installation." [Online]. Available: http://confine-orm.readthedocs.org/en/latest/#installation

[69] "The management network, Confine wiki." [Online]. Available: http://wiki.confine-project.eu/arch:management-network

[70] "tinc wiki." [Online]. Available: http://www.tinc-vpn.org/

[71] "IPv6 everywhere with tinc — Ivan. Loves. Gazpacho." [Online]. Available: https://elvil.net/drupal/en/post/ipv6-everywhere-with-tinc

[72] "CONFINE Wiki. Tinc Management Network overlay." [Online]. Available: http://wiki.confine-project.eu/soft:tinc, urldate = 26/04/14

[73] "Fed4FIRE. Get An Account and Certificate." [Online]. Available: http://doc.fed4fire.eu/getanaccount.html