Tools to foster a global federation of testbeds

Jordan Auge, Thierry Parmentelat, Nicolas Turro, Sandrine Avakian, Loïc Baron, Mohamed Amine Larabi, Mohammed Yasin Rahman, Timur Friedman, Serge Fdida

Tools to foster a global federation of testbeds

Jordan Augé[a,*], Thierry Parmentelat[b], Nicolas Turro[c], Sandrine Avakian[c], Loic Baron[d], Mohamed Amine Larabi[b], Mohammed Yasin Rahman[a], Timur Friedman[a], Serge Fdida[a]

[a] *Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France.*
[b] *INRIA, Sophia-Antipolis, France*
[c] *INRIA, Grenoble, France*
[d] *CNRS, LIP6 laboratory, Paris, France*

Abstract

A global federation of experimental facilities in computer networking is being built on the basis of a thin waist, the Slice-based Federation Architecture (SFA), for managing testbed resources in a secure and efficient way. Its success will depend on the existence of tools that allow testbeds to expose their local resources and users to browse and select the resources most appropriate for their experiments. This paper presents two such tools. First, SFAWrap, which makes it relatively easy for a testbed owner to provide an SFA interface for their testbed. Second, MySlice, a tool that allows experimenters to browse and reserve testbed resources via SFA, and that is extensible through a system of plug-ins. Together, these tools should lower the barriers to entry for testbed owners who wish to join the global federation.

Keywords: Future Internet; testbeds; tools; GUI; federation

1. Introduction

The internet is an old network, based upon protocols that were developed forty years ago. Its design choices were not made with current needs, such as mobility, wireless communications, real-time interactions, audio and video transmission, contemporary security concerns, etc., in mind. Applications such as the web, peer-to-peer communications, video streaming services, and many others were not anticipated. As we look to the future, with an explosion in the number of communicating agents using the network, and an increasing heterogeneity in the way the network is used, there will be a growing incoherence between the underlying technology and the uses to which it is put.

The shortcomings of the current internet architecture have prompted research initiatives to develop the so-called Future Internet, either through a continuation of incremental deployments or via a clean-slate approach based upon a complete rethinking of the internet's architecture. Testbeds have emerged from the need to explore this range of possibilities experimentally. They allow for experimentation on a multitude of existing and emerging technologies (sensor networks, wireless access, distributed computing clusters, switches, optical networking, etc.).

A great challenge in the development of these testbeds is to make them both highly capable and easily accessible to experimenters. The capabilities that we seek include great flexibility (for instance, to try radically new approaches at all layers of the protocol stack), global scale, an access to real end-users, high performance, and the ability to test cutting edge technologies. Such capable testbeds are complicated and expensive to put in place, and any given type of testbed might exist at only a few institutions, requiring experimenters to cross administrative boundaries in order to access them.

Because of scarcity, there is a need to share testbed resources among many different experimenters, granting each one what is termed a *slice* of the whole. Slicing can be achieved through sharing, either through virtualization, where this is possible; through other forms of simultaneous resource division (spatial, frequential); or through temporal division of resources, granting them to different users at different times.

Efforts are underway in Europe (the FIRE initiative), the United States (GENI [1]), and elsewhere in the world to promote the widespread availability of Future Internet testbeds through federation. In a federated system, experimenters at one institution can be authenticated and authorized to gain access to testbed resources hosted by other institutions. In addition, if physical interconnections exist between testbeds, a highly capable federation has the potential to allow experiments that span multiple testbeds at once.

*Corresponding author
Email addresses: jordan.auge@lip6.fr (Jordan Augé), thierry.parmentelat@inria.fr (Thierry Parmentelat), nicolas.turro@inrialpes.fr (Nicolas Turro), sandrine.avakian@inria.fr (Sandrine Avakian), loic.baron@lip6.fr (Loic Baron), mohamed.larabi@inria.fr (Mohamed Amine Larabi), mohammed-yasin.rahman@lip6.fr (Mohammed Yasin Rahman), timur.friedman@upmc.fr (Timur Friedman), serge.fdida@lip6.fr (Serge Fdida)

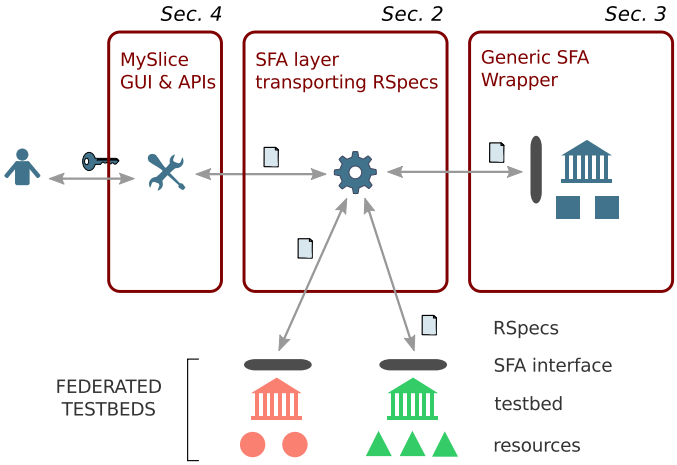

Figure 1: Positioning of the MySlice and SFAWrap tools presented in this article, illustrated by an experimenter browsing the list of available resources provided by an SFA-based federated testbed environment.

The de facto standard for testbed federation today is the *Slice-based Federation Architecture*, or SFA. A testbed owner who wishes to enter their testbed into the global federation needs to provide that testbed with an SFA interface. However, this is not necessarily an easy task. Several variants of SFA exist, both in the form of working code and written specifications [2, 3, 4]. Some aspects of SFA, such as its authentication mechanisms, require specialized knowledge to implement. Other aspects, such as parsing capabilities for resource descriptions, can be laborious to put in place.

The contribution of this paper is the introduction of two tools, SFAWrap and MySlice, that lower the barriers to entry for testbed owners who wish to join the global federation. SFAWrap enables an owner to easily provide an SFA interface for their testbed, and MySlice is a tool for experimenters that accepts plug-ins that understand the semantics of individual testbeds. Both tools follow an open community development model and are released under free and opensource licenses.

Figure 1 shows both tools in relationship to the experimenter (on the left), the testbed and its resources (on the right), SFA (in the middle), and other federated testbeds (below). Section references in the figure describe the organization of this paper. Sec. 2 provides background on SFA. Sec. 3 describes SFAWrap. And Sec. 4 describes MySlice. Sec. 5, not shown on the figure, draws conclusions and points to future work.

The technologies and tools introduced in this article have a broad range of applications. We are specifically using them as the building blocks of the OneLab experimental facility [5], a federation of major testbeds for experimentation in computer networking.

2. The Slice-Based Federation Architecture (SFA)

To set the context for understanding the two major contributions of this paper, the SFAWrap and MySlice tools, this section describes the Slice-based Federation Architecture (SFA) that they support.

2.1. Overview

SFA has been designed to provide a minimal set of functionalities, a 'thin waist', that a testbed can implement in order to enter into a global federation. An experimenter in an SFA-based environment can transparently browse resources on any federated testbed, and allocate and reserve those resources.

Because of the potential for a very large number of testbeds, a global federation architecture faces a serious scalability issue. SFA introduces a fully distributed solution in which each peer testbed serves as the authority of reference for the resources that it brings, and each user community, along with its experiments, is represented by an authority (possibly, but not necessarily identified with an individual testbed). Authorities agree to recognize each other. By eliminating any requirement for a central component, beyond an agreed naming hierarchy, SFA scales with hardly any practical limit.

Under the SFA architecture, there is a separation between what is generic and what is testbed-specific. Testbed-specific information is captured in a resource model, called a resource specification (RSpec), which is a XML file transported by the SFA layer. SFA itself does not cover such aspects as resource models, policies, reservations or measurements. These are implemented on top of SFA. The following subsections detail important aspects of SFA.

2.2. History

The first federation of computer networking testbeds was set up between Princeton's PlanetLab Central and UPMC's and INRIA's PlanetLab Europe, starting in 2006, as part of the European Union's (EU) OneLab project, prior to the start of the FIRE or GENI initiatives. This initial federation was based on the pragmatic solution of synchronizing the central databases of each of the federated entities. In this way, users of each testbed gained full access to the resources of the other. This solution worked in the context of two peers, but was clearly not scalable.

Larry Peterson, of Princeton University, together with others who were preparing the GENI initiative, conceived of SFA as a way forward, for PlanetLab and networking testbeds in general. Developers at Princeton, together with Thierry Parmentelat at INRIA, created the first working deployment of SFA code. Starting in 2008, SFA was used to extend PlanetLab federation to other peers, such as PlanetLab Japan and EmanicsLab. Development work on the European side was supported by the newly created FIRE initiative through the OneLab2 project. Simultaneously, SFA was adopted as a control

plane architecture by GENI, in which context written specifications [2] were drafted. (In addition to the authors of this draft, Jay Lepreau and John Wroclawski contributed to defining SFA.)

This article's description of SFA draws upon both the working code (from PlanetLab and from other, more recent, SFA implementations) and the written specification. These differ somewhat in their details, but agree on most of the main aspects.

2.3. SFA design principles

SFA is designed around a set of base objects:

Authorities: These represent testbeds, parts of testbeds to which trust or rights may be delegated, and/or communities of users.

Resources: These consist of nodes, links, or any other experimental resource exposed to the users.

Users: These are experimenters wanting access to resources.

Slices: A slice is the basic unit of interaction between users and resources. One can think of a slice as corresponding to an experiment, and encompassing all users and resources associated with that experiment.

SFA defines a set of functionalities to manage the different entities involved in a federation. The remainder of this section describes these functionalities, as well as how they are abstracted into various managers.

2.3.1. Naming

In SFA, each object is uniquely identified by a (type, HRN) pair, where the HRN is a Human-Readable Name. They are stored as records in a database, called the *registry*, alongside associated information. The *registry manager* is a component dedicated to the manipulation of this registry: creating, displaying, updating, deleting and listing objects.

SFA objects exist in a shared namespace, organized according to a hierarchy of authorities and sub-authorities. Trust relationships are based upon this hierarchy, with authorities vouching for the objects further down in the hierarchy. Figure 2 represents a part of the object hierarchy in the PlanetLab federation. We see the PlanetLab Europe (ple) and PlanetLab Central (plc) root authorities. We also see the sub-authorities, such as upmc and inria, princeton and columbia, that represent PlanetLab sites, which are the different universities and research labs that contribute resources to the testbed. These sub-authorities delegate the management of their own users to the root authorities. The figure highlights a user, represented by the HRN `ple.upmc.userA`, who has been vetted by the UPMC site under the PlanetLab Europe root authority. A registry is responsible for managing all of the objects for a given authority. It can delegate management to registries further down in the hierarchy. It will either handle requests itself, or route them to the responsible authority.

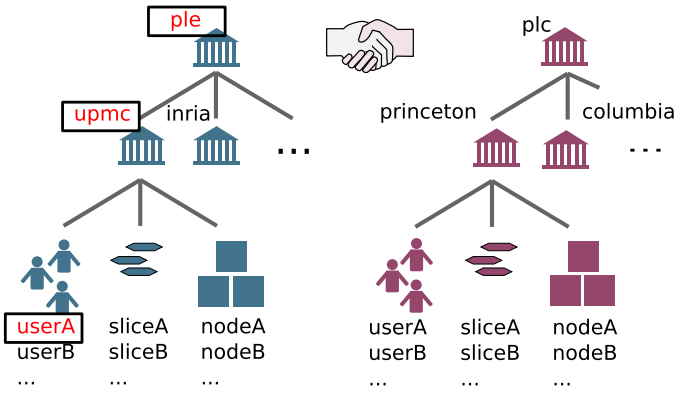

Figure 2: Hierarchy of objects and their naming, highlighting the naming of ple.upmc.userA in the context of federation between PlanetLab Europe (ple) and PlanetLab Central (plc).

2.4. Authentication, authorization, accounting and trust

SFA's authentication mechanism is based on a public key infrastructure, where each object has a keypair (a public and a private key) and is associated with a signed certificate, called a GID, that is stored in a registry. (Awkwardly, for a global federation scheme that is based on the notion that there is no one central authority, the term GID is said in some documents to stand for *GENI identifier*, perhaps *Global identifier* would be a better name.) The certificate is used for authentication, following the same principles as user and website authentication on the web. It is an X.509 certificate that associates the object's HRN with its public key, and that is signed recursively by each parent authority up to the root.

A user (U) can bootstrap the process by generating a self-signed certificate from its keypair, and getting it signed by its home authority (H) to form a GID. For H to be able to recognize U as its own user, a preliminary registration procedure is necessary so that H knows U's public key. U proves its identify by ciphering the communication with H with his private key thanks to the SSL protocol. U is then able to get authenticated to a third-party authority (A) via the same mechanism, but using the GID instead of the self-signed certificate. Even though A does not know U's public key, if U's root authority (H) is trusted by A, the chain of signatures in the GID is trusted as proof that the owner of this keypair is really U. In addition to users, authorities and resources can be authenticated using the same process.

SFA separates authentication from authorization. The latter depends on the local policies in use on the various testbeds, to determine which resources can be accessed, or which actions can be performed. An authorization credential is a signed XML document, that proves that an entity has a set of rights and states whether or not it has the possibility to delegate those rights. Such credentials can be used to establish the various trust relationships necessary to run a federated platform. For example, a slice credential might allow a privileged user to create a slice, while

a less privileged user might only be allowed to perform operations on an existing slice.

There is currently a debate in the SFA community as to whether to move to an attribute-based access control (ABAC) authorization mechanism, in which a user could assemble a set of signed clauses from various entities, and use them to construct a proof that they (the user) indeed have the rights that they claim. Shibboleth [6], which is used to manage a federation of identity providers for national research networks (NRENs), is also a candidate for a future authorization system.

Because authentication requires the possession of a private key, a delegation mechanism has been implemented in SFA so that a user could perform actions for which they have been delegated the rights, on behalf of another user. A delegated credential states that the delegee has some rights on an entity and is signed by the delegating user. It also encloses the original credential, proving that the original user has both the delegated privileges and the right to delegate them. Sec. 4 describes how this mechanism can be used to perform authentication and authorization by a remotely hosted tool.

2.5. Common API

SFA defines a minimal set of API calls to enable interaction between the different actors of the federation. The API calls can be grouped into three main categories:

Object management: These calls manipulate registry objects through the classical list, create, read, update and delete functions.

Resource browsing and slice management: These calls associate resources to slices, as well as starting, stopping or getting the status of slices.

Federation discovery: There is an API call that is used to obtain detailed information about the different federation services that are running, and to recursively discover peer platforms.

Different components that are used in the federation can provide different subsets of this API, depending upon their role. Three commonly deployed components are:

Registry manager (R): This accepts API calls to manage authorities, users, slices, and resources. A registry would typically be run by an authority to manage all of these items.

Aggregate manager (AM): This accepts API calls to browse and reserve resources. An AM would typically be deployed by an individual testbed.

Slice manager (SM): This is a convenience service that exposes the same set of API calls as an AM, but across multiple testbeds. A user would typically connect to an SM to get access to a range of federated testbeds.

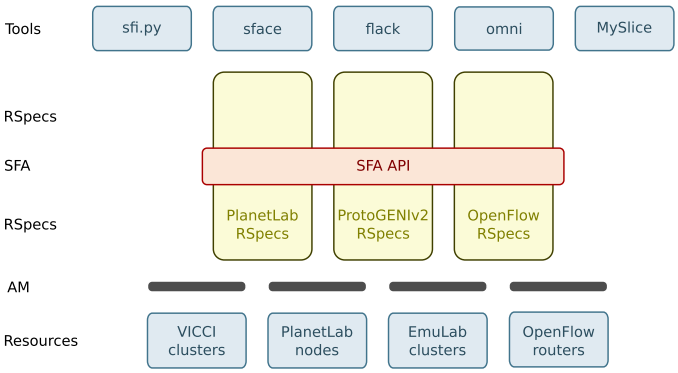

Figure 3: The SFA hourglass: SFA and RSpec provide the glue between heterogeneous testbeds and user tools.

SFA is based on a web services API. To issue a call, a user must connect to a manager's XML-RPC interface via HTTPS, using their private key as a cypher, and passing as a first parameter the credential that shows that they are authorized to perform the operation.

2.6. Resource model

Because it is impossible to predefine the different resources that will be made available by all testbeds, SFA has been designed to abstract everything that is testbed specific into a resource specification, or *RSpec*, an XML file that is transported over SFA. Fig. 3 illustrates how SFA and RSpecs provide the glue between heterogeneous testbeds and user tools. SFA is agnostic to the choosen information model while user tools and aggregate managers must be aware of RSpec formats.

2.7. Joining the federation

In order for a testbed to become part of the current global federation enabled through SFA, given that a trust relationship has been established with at least one current member of the federation, there are two important technical requirements that need to be fulfilled. First, the local testbed resources must be described in an RSpec that the testbed's aggregate manager can both send and understand. Second, a friendly user interface must be available for researchers to be able to browse the available resources, express their requirements, and reserve the desired set of resources of this testbed in a fashion that is consistent with the rest of the federation.

3. SFAWrap

3.1. Motivations

The aim of both SFAWrap, discussed in this section, and MySlice, discussed in Section 4, is to lower the barriers to entry into the global testbed federation for testbed owners. These owners know their equipment, they know how to describe it, the ways in which it can be reserved

and provisioned, the types of experiments that their testbeds can support. They should be allowed to focus their development efforts on the aspects of the control and experimental planes that directly relate to these issues, and leave other issues to be handled by others.

In achieving this aim, SFAWrap handles the server side, while MySlice handles the client side. On the server, which is to say the testbed, side, an owner may find themself in one of three possible situations: (1) they are building a testbed from scratch, (2) they have an existing testbed, but it is not yet federated via SFA, or (3) they have a testbed that already offers an SFA interface. SFAWrap is designed primarily to ease federation for owners in the first two situations, but it can potentially be of use to those in the third situation.

SFAWrap is written in Python. If someone is building a testbed from scratch, they can extend the tool's API calls in order to manage their testbed. A cleaner architecture would be to write their own testbed API (for example based on a RPC interface such as REST or XML-RPC), and have SFAWrap access the testbed through it. In this setup, the testbed's API can use any semantics that the testbed owner desires and it can be implemented in any language, on any operating system, using any database technology.

For the owner of a testbed that does not yet speak SFA, making their testbed SFA-compliant would be somewhat daunting without SFAWrap. They would first need to understand SFA, which, at the time of writing, is a moving target. Written specifications exist, but working code implementations do not necessarily adhere to them in all respects. And the working implementations diverge from each other. Delve into the code, and you might find stubs for procedures that are intended to be developed one day, or perhaps are legacy items that have been abandoned. Those in the United States can refer to a GENI reference implementation, but there is no international consensus to adopt this version rather than another.

Even if the owner does gain a good understanding of SFA, there are elements that are complicated or laborious to develop. The code that handles authentication and authorization is a notable example. The cryptographic basis for this functionality is a specialized area, well outside of most testbed domain specific expertise. It is true that one can use libraries that have built in functions for handling keys and certificates. Still, mastering these libraries and debugging an implementation that is based upon one of them, is a nontrivial task. RSpec generation and parsing is another example. Of course any competent programmer can write code to do this, but it can be laborious.

Rather than writing their own code, it is easier for a testbed owner to use existing code that already handles SFA-based authentication and authorization, that already has built in capacities for generating and parsing RSpecs, and that handles other aspects of the SFA interface. The benefit is not simply a one-off; since SFA is evolving, testbed interfaces will likely need to be recoded several times

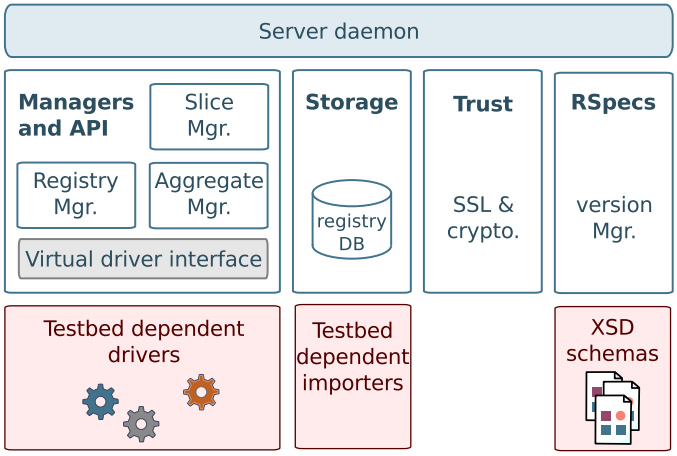

Figure 4: SFAWrap architecture

before the specifications settle into a standard. Recognizing this, an owner of a testbed with a native SFA interface might conceivably wish to drop it in favor of SFAWrap.

3.2. The development of SFAWrap

SFAWrap's origins are in PlanetLab's SFA implementation which was developed jointly by Princeton University and INRIA, as described in Section 2.2. This code was developed as a wrapper around the PlanetLab API. Once the utility of a wrapper was established, it was clear that other testbeds could be wrapped in a similar way. UPMC's and INRIA's involvement in several projects aimed at heterogeneous testbed federation provided the motivation. Developers from the two institutions then refactored the PlanetLab SFA wrapper to create SFAWrap.

SFAWrap has a proven track record in supporting federated testbeds with large numbers of users, as it continues to be the PlanetLab wrapper. It is now being applied to IoTLAB (formerly SensLAB) [7] (wireless sensor nodes) in the context of the French F-Lab project [8], to FEDERICA [9] (routed IP circuits, servers and routers) in the EU's NOVI project [10], to NITOS [11] (OMF-based wireless mesh nodes) and FITeagle [12] (various resources) in the EU's OpenLab project [13], and to OFELIA [14] islands (diverse openflow components) in the FIBRE and FED4FIRE EU projects. Each of these wrappings involves completely different testbed architectures.

3.3. Architecture

Figure 4 portrays the overall architecture of SFAWrap. It consists of a server that can be specialized into either a registry, an aggregate manager, or a slice manager. The codebase is structured around the different functional blocks represented in the picture:

Managers and API: These modules implement the logic of the different managers. They differ mainly through the set of API methods that each one exposes through its XML-RPC interface.

Storage: The registry database stores the various SFA-level objects (authorities, resources, users, slices) and their properties. This data is accessible through the registry manager API. Since the testbed maintains its own testbed-specific database, testbed-dependent importer scripts are used to initialize and/or synchronize the SFAWrap database with the testbed's own database.

Trust: Trust mechanisms are at the core of SFAWrap, and make up most of its codebase. These modules implement the various objects (certificates, credentials, etc.) that are required for SFA's authentication and authorization mechanisms.

RSpecs: A resource specification (RSpec) describes the set of resources made available by a testbed. SFAWrap provides helper classes for the parsing and manipulation of these specifications. The structure of these XML files is specified by means of an XSD schema.

Communication with the wrapped testbed is ensured by the virtual driver interface. It abstracts the necessary interactions into a set of hooks that make up the structure of the testbed drivers. They consist basically in the translation between the requests coming from the SFA layer into native testbed commands or API calls.

A basic client library is also provided, to help users to configure their access to the SFA federation, and to bootstrap the necessary credentials. On top of it, the `sfi` command line client allows a simple interaction with the connected testbeds, and can for example be used for retrieving RSpecs. Helper scripts are also provided to edit RSpecs (marking a resource to be added to a slice, for instance), or to perform actions on the registry.

The colored boxes in Fig. 4 highlight the different places where a testbed owner will intervene to make their testbed SFA-compliant. These are the driver and importer script, as we have described. Furthermore, a new RSpec XSD schema might also prove necessary. The following subsection describes the steps required in order for a new testbed to use SFAWrap.

3.4. Steps to wrap a testbed

A series of four simple steps is all that are required to wrap a testbed with SFAWrap, allowing it to expose its resources to the global federation. A detailed tutorial can be found on the SFAWrap website, `http://www.sfawrap.info/`.

(1) Mapping and naming SFA objects. All that is required is to assign a unique prefix to the testbed, define a naming hierarchy, and name all the objects, as illustrated in Figure 2.

(2) Installation and configuration. If a testbed owner adopts an already supported API, then all they are required to do is implement that API on their testbed. If no existing API is suitable, but the testbed uses an already supported RSpec, then the only additional required step is to write a driver in Python to translate between SFA calls and calls that are native to the testbed's API. (For example, translating an SFA-based *list resources* call into testbed API calls for *list nodes* and *list links* calls, for a testbed that is structured into nodes and links.) Finally, if the testbed introduces a new RSpec then a final required step is to extend the RSpec generation and parsing module to support the features unique to that RSpec.

(3) RSpecs. For exposing resource information, either an existing RSpec will suffice, or it will be possible to propose a new one. A new RSpec will not be entirely new; it will reuse as much as possible structures that have already been defined in existing RSpecs. At the moment, there are two coexisting flavors of RSpec, that each have their own characteristics and that might eventually converge. SFAWrap supports them both. The so-called *SFA RSpec* is the original PlanetLab RSpec, extended to allow for other testbeds. Some high-level structures have started to be harmonized across testbeds, to allow different RSpecs to more easily be merged. The parts of this RSpec related to resource reservation are also being defined so that they work across testbeds. The *ProtoGENI RSpec* is based on a mechanism of schema extensions. Once the schema is defined, the user can extend the editing and parsing capabilities of the RSpec modules. Future work will focus on making RSpec format changes as transparent as possible to testbeds.

While there is still room for improvement, the recent move from Aggregate Manager API version 2 to version 3 illustrates the benefits of using SFAWrap, both testbed developers and users, since this could be handled almost transparently. No change was required on testbed drivers, and the clients could continue to use the same tools, either the commandline version, or those using the client library integrated into the software package.

(4) Federation. Once the SFA interface is ready, the final step is to create federation links to one or more authorities that provide credentials for users. This is done by installing the certificates of those authorities and configuring local policies for the degree of access to provide to each authority's users.

In today's testbed environment, we observe that authorities are tightly coupled to individual testbeds[1]. Thus, a federation link to another authority corresponds to a federation link to another testbed. (By 'link' here, we mean

[1]They might as well represent groups of users, for instance an institution in the PlanetLab Europe case, illustrated in Figure 2. Also, the OneLab Experimental Facility, presented at the end of the introduction, consists an authority which has only users, and no resources.

the formalization of a trust relationship, not a physical link.) A convenience function in SFAWrap allows one to list both the aggregate manager and the registry of federated peers. When trust relationships have been configured transitively, one can recursively discover the full set of federated testbeds.

3.5. The IoTLAB case

Wrapping the testbed. There are four IoTLAB testbeds [7] in France. Each is a large scale open wireless sensor networking platform. The IoTLAB control framework was based on a homemade portal, using LDAP to store its users and ssh keys. The reservation and effective sharing of resources among users is managed by the OAR [15] job scheduler tool, popular in the Grid community.

From the SFAWrap point of view, IoTLAB brought both a new platform architecture and new hardware with its own characteristics: finegrained localization information in the form of (x, y, z) coordinates that are important for the knowledge of the radio topology, a binary mobility property, and leases related to reservable nodes. The successful integration of this new platform into SFAWrap is a demonstration of the framework's efficiency and adaptability, and should help pave the way for the further integration of new testbeds.

The testbed driver for IoTLAB was developed by customizing an existing skeleton driver. The driver issues LDAP queries and interacts with the OAR reservation system through its RESTful API. The IoTLAB RSpec was developed by extending existing RSpecs to accommodate some new fields. Most of the development work has consisted in adding node reservation capabilities, as existing RSpecs, based upon virtualizable resources, did not allow for timeslot-based reservations to be expressed. Care was taken to discuss the matter extensively with other testbeds that had similar requirements so that the solutions could be generic enough to be written once and reused in different contexts.

Reservation capabilities. Exposing reservation information in SFAWrap consisted in addressing three different aspects: (1) developing a tool that is able to connect to SFAWrap and enable an experiment to manage reservations, (2) elaborating an RSpec format for transporting reservation information, and (3) creating the right interface on the SFAWrap side.

The first item is based on MySlice and will be described further in Section 4.4.2. RSpecs have been extended with a new section storing leases, that correspond to the timeslots granted to a slice. A lease is characterized by the slice identifier, a start time and a duration; it contains a list of references to the resources it holds. Note that the duration of the lease is necessarity a multiple of the reservation granularity of the different resources. Finally, the third aspect has been handled by an SFAWrap component that is able to exchange reservation information with OAR. The LDAP and OAR modules developed for wrapping a IoT-LAB testbed can be reused by any other testbed owner willing to integrate such functionalities. They can also serve as a model for bridging any other reservation tool to SFAWrap.

Discussion. In addition to being a pioneer in adopting SFAWrap, initially developed for PlanetLab, the IoTLAB testbed is the perfect illustration of the versatility of the tool since it was built on a completely different model of slices and experiments (the notion of leases), as well as a different technical architecture (LDAP and OAR). As such, it involved writing extensions to the 3 different modules mentioned in Figure 4. Experience gained from this process has allowed for several improvements, paving the way for the large number of different testbeds that have followed.

Finally, the community-driven model that has been chosen has fostered the collaboration between the different teams, the adoption of common solutions for such issues as resource reservation, and the sharing of best practises. Future envisioned extensions, such as mobility, will certainly deserve a similar approach.

3.6. Related tools

Besides SFAWrap, we know of only two examples of SFA interfaces that are meant to be generic, rather than testbed-specific. One is the reference implementation of the GENI Aggregate Manager API [3]. As we understand it, this implementation, written in Python, serves as a working example of the AM portion of the SFA API. (Associated code also implements a command-line SFA client called omni and another server-side API for an entity called the GENI Clearinghouse.) It can be used for interoperability testing, and code can be freely borrowed, but this code is not intended for wrapping testbeds, and is not used for that purpose.

The other example is the AMSoil component [16] developed within the European Union's OFELIA project. This is positioned like SFAWrap, to make it easy for a testbed to offer an SFA interface. We are not in a position to assess it relative to SFAWrap, as, to our knowledge, it has yet to be deployed to wrap a testbed.

4. MySlice

4.1. Motivations

As does SFAWrap, MySlice aims to lower the barriers to entry to the global testbed federation for testbed owners. Whereas SFAWrap handles the server side, MySlice handles the client side. Federated environments pose a challenge to testbeds, since tools that work across a federation do not necessarily adapt easily to heterogeneity. MySlice's architecture allows testbed owners to design plugins that expose the unique features of a facility to experimenters, thereby making it easier to bring that facility into federation.

Up until recently, each computer networking testbed had its own separate user community. In the most common case, the testbed owner would build the facility for themself, and so the community would consist of, say, the owner's students and engineers. Some more costly facilities have attracted experimenters from outside their home institutions. Notable examples are the ORBIT wireless testbed [17] at Rutgers University's WINLAB, the multiple Emulab instances [18], and the global PlanetLab federation [19]. But despite the communities being larger, each one is still very much tied to its respective testbed. If we want to enable one testbed's users to easily use other testbeds, providing them with a single familiar tool that works across platforms will help. MySlice is designed to be such a tool.

We have designed MySlice to meet a number of goals that we describe in the following subsections. These are: achieving a proper balance between the need to present experimenters with uniform interfaces and the need to expose to those experimenters the full richness of heterogeneous environments; allowing for the synthesis of multiple sources of data about a testbed, in some cases beyond what is provided by the testbed itself; hiding SFA's complexity from users; providing a graphical user interface for new users, combined with a web services interface for power users, and a programmable library-style interface for those who wish to incorporate MySlice's features into a larger tool; and ensuring the ease of development for testbed owners who are designing plug-ins. This section concludes with a brief account of how MySlice came to be developed.

The code for MySlice is available from the project website at `http://www.myslice.info`.

4.1.1. Balancing uniformity and heterogeneity

A cross-platform tool for experimenters faces conflicting imperatives: it should represent the full variety of heterogeneous environments to its users (the real value of the testbed), while at the same time offering those users a familiar interface that is homogeneous across environments. Any resolution of this conflict is necessarily a compromise. MySlice's system of plug-ins strikes a balance that combines ways of representing testbed resources that are common across all testbeds with the ability to represent features that are specific to an individual testbed or type of testbed.

To give an example of commonality, any set of similar testbed resources (a set of Wi-Fi nodes, say, or a set of virtual servers) can be represented in a table: each line lists a different resource and each column describes a characteristic of that resource (its ID, its location, how many interfaces it has, etc.) MySlice offers experimenters with a tabular view of testbed resources for all testbeds. Indeed, even when faced with an unfamiliar RSpec, for which all the semantics are not known, MySlice does its best to provide a tabular representation. When the semantics are known (for instance, that node reliability is a real value between zero and one), MySlice improves the representation to take them into account.

An example of heterogeneity arises with testbed node layout in Cartesian space. When selecting wireless nodes, location is very important; one typically wishes to have nodes that are relatively close to each other, to ensure connectivity. The experimenter should have access to the location in three-dimensional space of the nodes. For PlanetLab nodes, location is also important, but these are coordinates with the two dimensions of longitude and latitude. In comparison, when selecting nodes in an emulation testbed such as Emulab, the location of a given server in the rack is irrelevant.

As part of its effort in the F-Lab project [8] to make the IoTLAB sensor networking testbeds available to experimenters through MySlice, INRIA has developed a plug-in for the MySlice GUI that shows nodes in three-dimensional space. The visualisation can be rotated and nodes can be selected and deselected. This plug-in can be used for any IoTLAB testbed and indeed for any testbed that provides three dimensional coordinates for its nodes. For PlanetLab and other geographically distributed testbeds, there is a Google Maps plug-in, similar to what the Flack [20] tool offers. For testbeds in which location is not a factor, no such plug-in would be offered.

4.1.2. Combining multiple sources of data

As part of UPMC's experience operating PlanetLab Europe, we quickly became aware that users choose their nodes on the basis of features that are not communicated by MyPLC, the central PlanetLab server (the aggregate manager, or AM, in SFA parlance). PlanetLab provides a minimalist design [19], encouraging third parties to develop services that run on top of it. So even basic information about nodes, such as their load levels or uptime, are not known to MyPLC. A service called CoMon [21] measures such characteristics, and experimenters often draw upon CoMon data to choose nodes that are lightly loaded and reliable. Similarly, a measurement service that UPMC has developed, TopHat, relying upon the TopHat Dedicated Measurement Infrastructure (TDMI) [22] and others, provides network-related information about nodes, such as in which autonomous system they are located, and pairwise data, such as the number of traceroute hops between nodes. Experimenters who wish to choose widely spaced nodes can use this data in order to do so.

Prior to the development of MySlice, these sources of data were disparate, and experimenters needed to seek them out. In creating MySlice, we have aimed to synthesize multiple data sources for the user. We have described this approach in a previous paper [22], which focused exclusively on the TopHat system that does this for MySlice, so we will not go into detail on it here. The main idea, from the SFA perspective, is that an aggregate manager provides a basic RSpec for a testbed, which TopHat can then enrich through annotations from multiple measurement sources. For some testbeds, in which the aggregate

manager is omniscient, this feature would not be needed, but in the general case, and notably in the cases of open testbeds (such as PlanetLab) or experimenter-originated measurements, we believe it is useful.

4.1.3. Hiding complexity from users

Just as SFAWrap shelters testbed owners from the full complexity of implementing an SFA interface, MySlice has been conceived to hide some of the complexity of SFA from experimenters. In particular, keys, certificates and credentials can be tricky for users to manage, and MySlice aims to manage these on users' behalf.

A user's private key should typically be stored in their own account, preferably on a machine that they themselves own. Accordingly, one would think that we could hide the complexity of key and certificate management by placing an SFA client tool on the user's machine as well. But if we do this, we run into the difficulty of developing a tool that works across a variety of operating systems. There are solutions, notably command-line tools based upon widely-available languages such as Python or Ruby, or graphical interfaces based on Java or Qt. For the greatest cross-platform functionality, however, providing both graphical and programmatic interfaces, a web-based tool is ideal.

MySlice aims to provide the ease of use of a web-based service while at the same time sheltering experimenters from the fussier aspects of the authentication and authorization schemes. To do so, it uses SFA's delegation feature, allowing the experimenter to keep their private key private by delegating their authority to MySlice to carry out actions on their behalf. Another proposed option a user has is to trust the tool for fully managing his account, which might prove even more simple for him. In this latter case, the tool will continue to use delegation for accountability purposes with respect to the testbeds.

4.1.4. Providing a variety of user interfaces

Experimenters have a variety of styles for carrying out their work. Some, typically new users, will want a nice GUI through which to access testbeds. Others will want the flexibility and power of a programmatic interface. MySlice has been conceived to cater to both types of experimenters, as well as providing a smooth transition path for one type of user to become the other. Each time that an experimenter uses MySlice GUI, they have the option of seeing a snippet of code that will allow them to carry out the same operation via XML-RPC. Fig. 7 shows an example of this: the query code displayed at the bottom of the page provides the data that is shown in the table further up the page.

In addition to providing interfaces for experimenters, MySlice provides a Python library for programmers of experiment control tools. This allows other, more powerful, tools to be built on top of the functionality provided by MySlice.

4.1.5. Ensuring ease of development

Separate from the interfaces mentioned in the previous subsection is the interface presented to developers of MySlice plug-ins. We wish to make it as easy as possible to create new plug-ins such as the three-dimensional coordinate visualisation one mentioned above, a reservations plug-in that is under development, and many others. We therefore based plug-in development on the most commonly found web GUI development skills: JavaScript and Python/PHP programming. An alternative would have been Flash, as used by the Flack tool [20], however we were concerned by the lack of Flash support in certain environments (such as iOS), and also the fact that Flash development skills are not so widespread.

4.1.6. The development of MySlice

MySlice has its origins in MyPLC [19], the central server for PlanetLab systems. As we have described in earlier work on the TopHat measurement aggregation system [22], we started by enriching MyPLC's user interface with measured characteristics of the nodes. Because of the large number of measured features, this required a flexible manner of presenting tabular data. This development was joint work between UPMC, Princeton University and The University of Tokyo, funded by the GENI Understanding Federation grant.

Once a flexible manner of presenting the data had been developed, it was a simple step to furnish data for systems beyond PlanetLab. We extended the collaboration in the French ANR's F-Lab project [8] to encompass the IoTLAB testbeds [7]. In this context, INRIA has developed the three dimensional visualisation plug-in and is developing the reservation plug-in.

4.2. MySlice architecture and design decisions

For reasons described above, MySlice provides two interfaces for experimenters: the web-based GUI and the programmatic XML-RPC API. In addition, it provides a library in the form of a Python API for people who wish to build tools on top of MySlice. These three interfaces are supported by three separate components, illustrated on Figure 5: the core functionalities are provided through a Python library, which in turn is exposed through an XML-RPC API for remote use, and by the web GUI.

The separation into separate components allows us to better target different developer communities. The choice of Python for the core is justified by its simplicity and efficiency for these more advanced tasks, as well as the large community of Python developers and available Python modules. The web interface is built with two flavours on top of the widely-used Joomla and Django frameworks, both allowing pages to be edited by non-specialized users. The development is done respectively in PHP or Python for the static parts, and JavaScript/jQuery for dynamic parts, two technologies fitting the community of web developers and designers. Both are quite standard and easily available for deploying the different components.

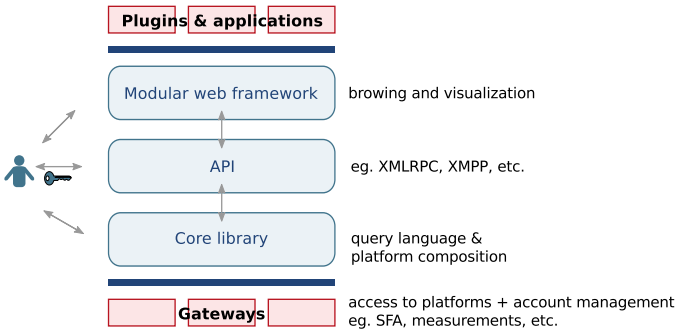

Figure 5: MySlice architecture

Behind the API, the library handles low-level interactions with SFA-based entities (aggregate managers, slice managers, and registries). Since SFA-based communication puts some common requirements on both the server side and the client side, MySlice is able to reuse code from SFAWrap. For RSpecs, MySlice borrows helpers that allow it to parse and edit RSpecs through a uniform API. For SFA authentication, it borrows code that allows users either to fully trust MySlice or delegate their rights to it. Also supporting the API is code for persistent local storage and caching of information, which improves request efficiency and thus the user experience.

The web interface builds on this API and is dedicated to providing an intuitive and efficient interface that allows users to browse available resources and pick the most appropriate ones for their experiments. It provides experimenters with information about these resources, enriched by multiple measurement sources if available, and allows them to select, filter, and visualise this information.

4.3. The paradigm underneath MySlice

In order to understand MySlice versatility, we need to give some insights into the data framework it builds upon, denoted MANIFOLD, which will be described more thoroughly in subsequent work. MySlice fully inherits its architecture from this component; Figure 5 is annotated on the right with the main functionality provided by each layer.

Data representation. MANIFOLD has been designed for interconnecting heterogeneous data providers. It assumes a uniform representation of data in a tabular format similar to what is found in relational databases, and whose flexibility has been much assessed in the literature. This representation is a fundamental aspect of the framework which makes it feasible and scalable. In particular, it assumes that all data can be identified through an appropriate naming (and thus that an underlying semantic has been adopted). The software does not need to know about the naming schemes to transport and process the data; it is thus evolutive and independent to evolutions of the data.

Let's take the example of RSpecs to illustrate how information can be mapped. In SFA, RSpecs serves the purpose of exposing detailed information about resource objects, holding several properties such as the unique name of the resource, its type, the responsible authority, etc. We can easily see the same information could be stored inside a database, eventually creating references to other relations for more complex data types. While RSpecs are loosely defined, in practice they all tend to follow a similar top-level structure, and only differ through some resource attributes. Simple heuristics have been proved sufficient in most cases. Furthermore, for testbeds wrapped with SFAWrap, the parsing code can be reused and it produces an appropriate structure. Finally, in the most extreme cases, a manual intervention is still possible with a few lines of code.

Platform adaptation. As existing platforms do not necessarily stick to this representation scheme, it allows for gateways to be developed, playing to role of an adaptation layer abstracting the heterogeneity in technologies (and eventually semantics). The SFA gateway is an example of such an adapter. It consists in an advanced client for a SFA aggregate or registry manager. The framework natively supports multiple users and is able to provide support for managing users' accounts on the different platforms. The handling of SFA authentication and authorization tokens introduced in Section 4.1.3 is performed by the SFA gateways as an instance of such functionality.

Query language. As part of its core library, MANIFOLD proposes a simple query language, largely inspired from SQL, allowing to request information aggregated from several platforms, independenly of the format of the data,or the technology behind.

Platforms just need to advertise the data they can provide, and the framework is able to make the necessary glu to provide connectivity to all of them. Just like SQL does not depend on the underlying database schema, our query language makes it possible to address all advertised information with a fixed set of API calls.

Data composition. Beyond the gateway, the component does not require any knowledge of this underlying semantic to transport and process the data. We nevertheless encourange platforms to agree and share at least parts of the semantics – through the use of ontologies for instance. This will enable the component to transparently infer the relationships in the data, and to compose the different data sources. This architecture thus makes it possible to enhance the value of existing platforms well beyond the value of each platform considered individually, answering queries overlapping two or more sources of information.

Building on this framework. The component advertises available aggregated data as if it were a platform itself. This allows the web interface to build on top of it to allow users to browse, navigate and make sense of the vast amount of available information. This is performed

Figure 6: Screenshot of the MySlice dashboard

through a set of visualization plugins or widgets guiding users through some process. Our experience operating PlanetLab Europe shows that such an interface is fundamental for users to exploit the full richness of the tool.

The framework is designed to be easily extensible via its plugin mechanism, allowing both new parts of the interface to be added, or even more complete applications or workflows. The portal functionality described later is such an example. Plugins benefit from several modules making development easier such as full support for asynchronous queries. The abstraction framework we have just described allow them to be isolated from the heterogeneity of the different interconnected platforms: they just need to issue queries with the right semantic to be put in relation with the right platforms and receive requested data.

As a general principle, the layering of functionalities makes it possible to use them separately. The SFA client embedded in the gateway can for example be used independently, and the same holds for plugins.

The reader is invited to refer to the project website for further technical insights into the component. The rest of this section provides greater detail on the MySlice web interface that has been built above this framework, describing the ways in which it has been designed to meet the needs of both users and testbed owners.

4.4. The MySlice web GUI

This subsection describes the functionality that is available to experimenters through the MySlice GUI. This includes both generic functionalities and extended and specialized functionalities that are available through plug-ins.

Services provided by the federation are typically centered around a single functionality (control plane, measurements, policy enforcement, etc.). At the different steps of their experiments though, users will have to contact several such services. For instance, while browsing resources, they might be interested both in the resources themselves (from SFA) and in measurements originating from a third party service, or even a stitching service to discover inter-testbed connectivity. This GUI will make it possible to assemble user-oriented interfaces, answering a specific need, and transparently involving the different services thanks to the composition capabilities of the framework.

4.4.1. MySlice portal and dashboard

The first page users will be faced with is the MySlice portal, allowing them to register and login, to create slices, and to browse and reserve testbed resources. For some of the more complex tasks, users will be guided through a series of simple steps. During registration for example, depending on their choices, they will be invited to generate a keypair that will be used for authentication to the testbeds, or simply upload their public key and delegate some of their rights to MySlice, as explained before. More details about the portal functionality can be found in [23].

Once a user has been authenticated by MySlice and authorized by a testbed, he can access testbed's resources. The MySlice dashboard (Fig. 6) is the page that welcomes users upon login. It provides an overview of the user's account and authorization credentials, the various types of resources that the user can access through the different federated authorities, as well as a list of the slices with which the user is associated. To generate this page, MySlice issues a series of queries to retrieve information about the different SFA objects related to the user, as presented in Sec. 2: users, authorities and slices.

4.4.2. The slice page and plug-ins

By clicking on the name of a slice in the dashboard, the user accesses the slice page (Fig. 7). This page is the heart of user interaction for the MySlice GUI, allowing an experimenter to learn about available resources, to sort and filter resources based upon their properties, and to associate selected resources with their slice.

The slice page is where the plug-ins are displayed. The page receives objects by issuing a query to SFA through the API, for which the reply is recursively displayed. If a given object (a slice, a list of resources, etc.) has a defined plug-in, then it is displayed using that plug-in. Otherwise a generic display is used.

Visualization. Different representations of the resources are possible. The simplest one is a table where each line represents a resource, and each column a property of the resource. When latitude and longitude information is present, the slice page offers a Google Maps plug-in. As mentioned earlier, this helps an experimenter to exploit the geographic diversity of a platform like PlanetLab.

Three-dimensional layout plug-in. In the context of the F-Lab project, in order to federate the IoTLAB testbeds, we developed the three-dimensional layout plug-in already mentioned. Fig. 8 shows a screenshot. It portrays the wireless sensor nodes of a IoTLAB platform, and allows the user to rotate the layout as desired and to select and deselect nodes. Though developed for IoTLAB, this plug-in could easily be applied to any other testbed in which three-dimensional coordinates are important.

Reservations plug-in. An effort currently underway is to display reservation information for resources that cannot

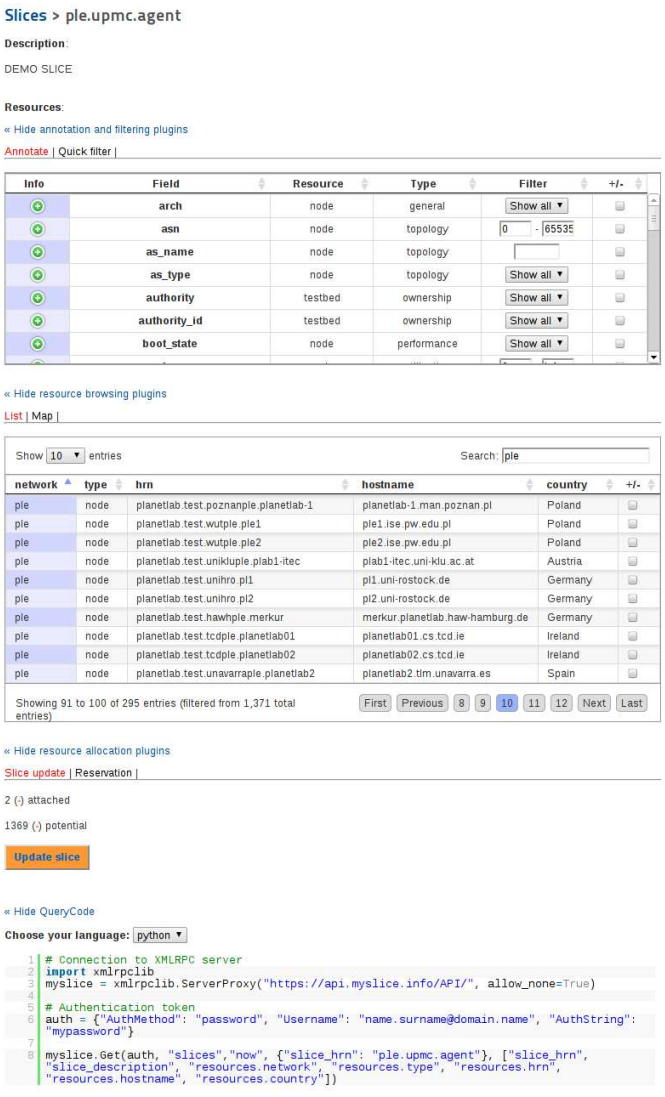

Figure 7: Screenshot of the slice page

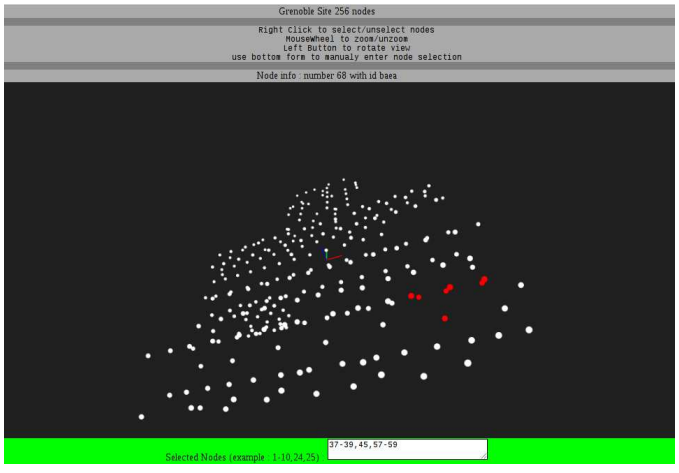

Figure 8: Screenshot of the 3D layout plug-in

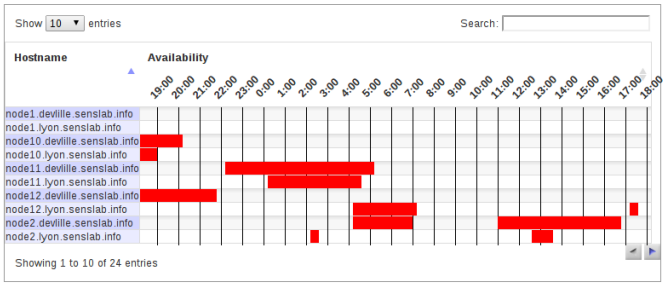

Figure 9: Screenshot of the scheduler plug-in

be virtualized or that need to be shared in time. Fig. 9 shows a screenshot.

A publish-subscribe architecture. In practice, there will be several possible representations of the data on the page. They are all kept synchronized through a publish-subscribe architecture, where different plug-ins subscribe to updates on the results of a query.

When there are large numbers of resources, experimenters will need to define filters in order to reduce the number of possibilities to a manageable number. The same pub/sub mechanism makes it possible to have several concurrent plug-ins editing the same or different parts of a query.

Additional features. A good GUI will not make users wait before showing data. Indeed, it is often better to display slightly out-of-date information quickly and then update it as soon as possible. The slice page does just this, drawing from its cache in order to display show data rapidly, then, based upon timeliness information associated with the data, launching queries for any information that might be out of date.

In order to help experimenters transition from the web-based GUI to a programmatic interface, the slice page includes a plug-in that displays the XML-RPC query that allows one to retrieve the results presented on the current page. Since such queries can be written in various languages, the plug-in offers a choice of languages so that the user can see the one that they prefer to use. For instance, the query shown at the bottom of Fig. 7 is in Python because that language has been chosen from the pull-down menu. We hope this feature will help create more power users: after navigating from the entry points provided by the web interface to some information of interest, they will be able to discover the related semantics, and how to programmatically access the same information. A simple copy-paste will provide a working example.

Finally, since all plug-ins are based on standard API queries, these queries can be easily logged. We intend to develop a plug-in that will monitor the way in which experimenters make use of the system. This will help us to answer such questions as which resources users value and for what reasons, leading to better testbed design.

4.5. Supporting new testbeds

MySlice is easily extended to support new testbeds. If SFAWrap was used to wrap the new testbed, then there will be parser code for its RSpec, either because the code already existed or because it was written as part of the wrapping process. MySlice can reuse that code. Otherwise, the testbed owner needs to write new parsing code, which they can do based on existing examples.

Once it can parse an RSpec, the semantics might be unfamiliar to MySlice. In that case, MySlice can display unknown resource properties in a best effort manner in tabular format. A new plug-in, or extension of an existing plug-in, is required in order to have a more sophisticated display. By fostering a community of contributors of free, open-source plug-ins to MySlice, at the MySlice web site (http://www.myslice.info), we make it possible for a developer to work from previous examples. The goal here is to give as much control as possible to the owners and users of testbeds, so that they can create interfaces that best match their needs.

4.6. Related tools

As mentioned above, there are other SFA client tools. The simplest one is *sfi* [24], a command line client that allows communication with SFA entities and that also provides a set of command line tools for editing RSpecs that respect known formats. The *Sface* [25] interface builds on sfi to provides a simple Qt-based GUI that displays information in tabular format. *Omni* [26] is a command line client that is known to work across multiple control frameworks, including different flavors of SFA.

Both OMF-F [27] and FSToolkit [28] present domain specific languages allowing users to write a program describing experiments, and requesting services and resources.

The only tool besides MySlice to go beyond the command line or a basic graphical interface is *Flack* [20]. It is capable of processing a variety of RSpecs and is able to represent resources on a world map. It allows users to create slices by drawing nodes and links on a canvas. Flack is written in Flash and is accessed over the web and runs on the local host.

MySlice and Flack are similar in that they are both open source clients that aim to provide a rich and user-friendly interface, easily apprehendable by beginners. Both clients abstract out the complexity of SFA and the diversity of RSpecs to propose a unified interface.

There are, however, important differences between MySlice and Flack: (1) while Flack requires a certificate and a private key, MySlice allows simpler authentication schemes. This is made possible by permitting experimenters to delegate their credentials to MySlice; (2) Flack has a monolithic design that is more complex to extend than MySlice. MySlice has been conceived around a modular architecture, offering different interfaces that are adapted to different communities of developers and users.

We note that Flack currently offers an interesting GUI for visualizing and editing network topologies. The plugin system of MySlice has been designed so that such capabilities are not a limit of the tool, but can be built as extensions to the existing framework.

5. Conclusion and future work

We are at a point in time when SFA has emerged as a de facto global standard for the federation of Future Internet testbeds. However, for a testbed owner, providing one's testbed with an SFA interface can be a daunting task. This is due to the complexity of some features (authentication, parsing) and the still fluid nature of the SFA specification, which will require regular code updates for any implementation that needs to remain current.

Our objective is to facilitate the participation of new testbeds in the global federation. For this, we have designed and put into practice two tools that allow a testbed owner to expose their local resources through the SFA API and that provide experimenters with interfaces suitable to the particular characteristics of each testbed. The modular design of our tools and the open source development paradigm that they follow create a framework in which testbed owners worldwide can benefit from drivers and plug-ins developed by others, and to which they can contribute in turn.

Today, we are in the process of testing the application of these tools to a small selection of existing testbeds including SensLAB and FEDERICA. In the near-term future, we plan to deploy scheduling capabilities, which will allow experimenters to reserve non-virtualizable resources. Over the longer term, we intend to roll out our tools for numerous testbeds that do not yet provide SFA interfaces.

Acknowledgments

The research leading to these results has received partial funding from a number of sources. The design and development of these technologies is funded by the European Commission's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 287581, OpenLab. The application of our tools to different testbed technologies is funded by the European Commission's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 257867, NOVI (for the FEDERICA testbed), and by the French ANR project F-Lab under grant agreement no. ANR-2010-VERS-002.

We are grateful to Panayotis Antoniadis for introducing the idea of MySlice and for implementing it as an internal service of the PlanetLab web interface with the help of Guthemberg Silvestre and Ahmed Arous at UPMC, and in collaboration with Andy Bavier and Tony Mack at Princeton and Aki Nakao at the University of Tokyo, to all of whom we are also grateful. Panayotis Antoniadis also helped with useful discussions during the preparation of this manuscript.

[1] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, I. Seskar, Geni: A federated testbed for innovative network experiments, Computer Networks – special issue on Future Internet Testbeds (????).
[2] L. Peterson, R. Ricci, A. Falk, J. Chase, Slice-based federation architecture (sfa), 2010. Working draft, Version 2.0.
[3] GENI AM API, http://groups.geni.net/geni/wiki/GAPI_AM_API, 2013.
[4] OpenSFA, http://www.opensfa.info, 2013.
[5] S. Fdida, T. Friedman, T. Parmentelat, OneLab: An open federated facility for experimentally driven future internet research, in: Proc. Workshop on New Architectures for Future Internet.
[6] R. Morgan, S. Cantor, S. Carmody, W. Hoehn, K. Klingenstein, Federated Security: The Shibboleth Approach, EDUCAUSE Quarterly 27 (2004) 12–17.
[7] SensLAB (project website), http://www.senslab.info, 2012.
[8] F-Lab: Federating Computing Resources (project website), http://www.f-lab.fr, 2012.
[9] M. Campanella, F. Farina, The federica infrastructure and experience, Computer Networks – special issue on Future Internet Testbeds (????).
[10] NOVI (project website), http://www.fp7-novi.eu, 2012.
[11] A.-C. Anadiotis, A. Apostolaras, D. Syrivelis, T. Korakis, L. Tassiulas, L. Rodriguez, M. Ott, A new slicing scheme for efficient use of wireless testbeds, in: Proceedings of the 4th ACM international workshop on Experimental evaluation and characterization, ACM, pp. 83–84.
[12] Teagle (project website), http://www.fire-teagle.info, 2012.
[13] OpenLab (project website), http://www.ict-openlab.eu, 2012.
[14] M. Su, L. Bergesio, H. Woesner, T. Rothe, A. Kpsel, D. Colle, B. Puype, D. Simeonidou, R. Nejabati, M. Channegowda, M. Kind, T. Dietz, A. Autenrieth, V. Kotronis, E. Salvadori, S. Salsano, M. Krner, S. Sharma, Design and implementation of the ofelia fp7 facility: The european openflow testbed, Computer Networks – special issue on Future Internet Testbeds (????).
[15] N. Capit, G. Da Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounie, P. Neyron, O. Richard, A batch scheduler with high level components, in: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2 - Volume 02, CCGRID '05, IEEE Computer Society, Washington, DC, USA, 2005, pp. 776–783.
[16] AMSoil, https://github.com/fp7-ofelia/AMsoil#readme, 2013.
[17] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, M. Singh, Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols, WCNC'05 (2005).
[18] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, A. Joglekar, An integrated experimental environment for distributed systems and networks, in: Proc. of the Fifth Symposium on Operating Systems Design and Implementation, USENIX Association, Boston, MA, pp. 255–270.
[19] L. Peterson, T. Anderson, D. Culler, T. Roscoe, A Blueprint for Introducing Disruptive Technology into the Internet, HotNets-I '02 (2002).
[20] Flack, http://protogeni.net/flack.html, 2012.
[21] K. Park, V. S. Pai, Comon: a mostly-scalable monitoring system for planetlab, SIGOPS Oper. Syst. Rev. 40 (2006) 65–74.
[22] T. Bourgeau, J. Augé, T. Friedman, Tophat: supporting experiments through measurement infrastructure federation, Proceedings of TridentCom'2010 (2010).
[23] L. Baron, J. Augé, T. Friedman, S. Fdida, Towards an integrated portal for networking testbed federation, an open platform approach, FIRE Engineering workshop, Ghent, Belgium (2012).
[24] SFA git repository, http://git.onelab.eu/?p=sfa.git, 2012.
[25] Sface git repository, http://git.onelab.eu/?p=sface.git, 2012.
[26] Omni, http://trac.gpolab.bbn.com/gcf/wiki/Omni, 2012.
[27] T. Rakotoarivelo, G. Jourjon, M. Ott, Designing and Orchestrating Reproducible Experiments on Federated Networking Testbeds, Technical Report, NICTA, 2012.
[28] FSToolkit, http://nam.ece.upatras.gr/fstoolkit/trac/, 2013.