A Walk Through the GENI Experiment Cycle

Thierry Rakotoarivelo, Guillaume Jourjon, Olivier Mehani, Max Ott, and Michael Zink

1 Introduction

One of the cornerstones of scientific discovery is validation by the community. In experimental science, this requires others to repeat the experiments and obtain similar results within acceptable statistical bounds. Traditionally, the distributed computing and networking community has been largely ignoring this. There are few publications in top-tier venues, which primarily report on the successful validation of somebody else's work, while problems with repeatability are sometimes buried in vague references. There are many reasons for that. Advances in the underlying technology continuously create new opportunities to explore new ideas leaving little time to reflect on the "old". But there are also very pragmatic reasons. First of all, most experiments are conducted in complex environments with many external dependencies, such as type and speed of computers and networks, size of storage, chip sets, or operating system and driver versions. Some of them will only affect the measured "utility" of the reported phenomena, while others are essential to having a successful experiment in the first place. Unfortunately, many of these dependencies are never reported and therefore making it very difficult for others to repeat an experiment.

We argue, that our inability in Computer Science to repeat reported experiments is not only bad practice, but also hampers progress in general. It reduces our ability to expand on prior work, verify and adapt it to different contexts, compare different methods in different environments and much more. We also argue that the "paper" as the traditional publication mechanism is one of the major obstacles in improving the status quo.

We are clearly not alone, initiatives, such as the Elsevier's Executable Paper Challenge [4] have been exploring new avenues for disseminating scientific res-

NICTA

Australian Technology Park, Eveleigh, NSW, Australia

e-mail: `firstname.lastname@nicta.com.au`
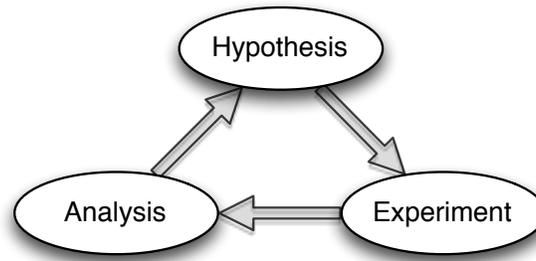
Fig. 1 Scientific Method.

ults. In addition, easy access to emerging large scale experimental facilities, funded and coordinated by programs, such as GENI in the US [1], FIRE in Europe [5], and similar activities in China, Korea, and Japan, provide the community with a common "playground" in which to conduct experiments. But only providing experimental facilities is not sufficient. The sharable resources we have available today still need to get provisioned, configured and modified before they can be used in experiments. We see those steps as the crucial pieces that are needed to perform repeatable experiments.

In the remainder of this chapter we propose and describe a family of tools to support an experimenter's workflow, while also automatically capturing most of the necessary information to allow others to repeat, inspect, validate and modify prior experiments.

More specifically, we propose to model the experimenter workflow on the Scientific Method[1] which we interpret, as shown in Figure 1, as a repeated cycle of stating a hypothesis, designing and conducting an experiment, and finally analyzing the measurements taken during the experiments with the intent to test or disprove the hypothesis.

We observed that many of these steps follow the same internal workflow of planning, preparing, and executing. We therefore built an experimenter-facing web-based tool, called LABWIKI, which supports this three-step workflow in different contexts. LABWIKI, as the name implies, is modeled after the traditional laboratory book, which experimenters use for a very similar workflow and purpose. LABWIKI takes this further, by not only being the recording mechanism, but also the operating platform for many activities within the experiment workflow.

We start the remainder of this chapter with a brief overview of LABWIKI (Section 2) and then introduce its user interface (Section 3). In Section 4, we introduce a real experiment used to research a time synchronization approach and published in [13], which will be used as a guiding example in the remainder of this chapter. An overview on experimental facilities and testbed resources is given in Section 5. We will illustrate the experiment workflow by taking the reader through every step, namely the experiment design (Section 6), the setup of and experiment (Sections 7

---

[1] https://en.wikipedia.org/wiki/Scientific_method
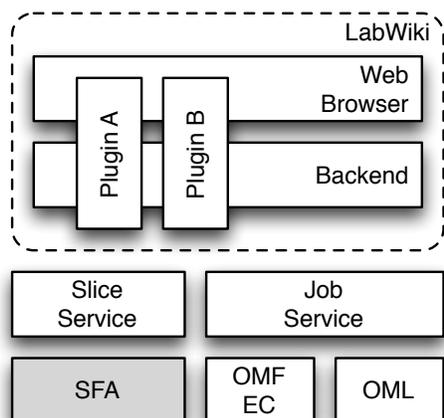
Fig. 2 LABWIKI and Supporting Services.

and 8), the execution of an experiment (Section 9), and the analysis (Section 10) of a previously published research result [13]. Finally, we briefly describe how the LABWIKI workspace can support educators in harnessing these large facilities for lab tutorials (Section 11).

2 LabWiki Overview

The web-based LABWIKI service strives to be the primary tool for an experimenter to plan, prepare, execute, analyse and even publish experimental-driven research. While the classical UNIX approach of "many little tools" often leads to a very rich and versatile environment it also requires great discipline on behalf of an experimenter to keep a detailed record of what combination of tools and their configurations have been used for what experimental artefact. On the other hand, a single comprehensive tool rarely works for cutting-edge research as requirements for new features often outstrip the development resources of the "mega tool" builder.

LABWIKI attempts to find a sweet spot by defining a framework which a) is based on well established, unifying methodology, b) supports the tracking of artefacts, their meta data and relationships to others, c) is extensible, and d) allows for easy integration of external tasks and services. Simplistically, it can be viewed as an easily customisable glue between the many little tools and the "history keeper" on how they were all used in the pursuit of a scientific discovery.

LABWIKI is the result of the shared experience of the authors in their respective roles as tool builders, testbed operators, researchers, educators, engineers, administrators and many more. The following is a description of LABWIKI's current "universe" of components, services, and capabilities.
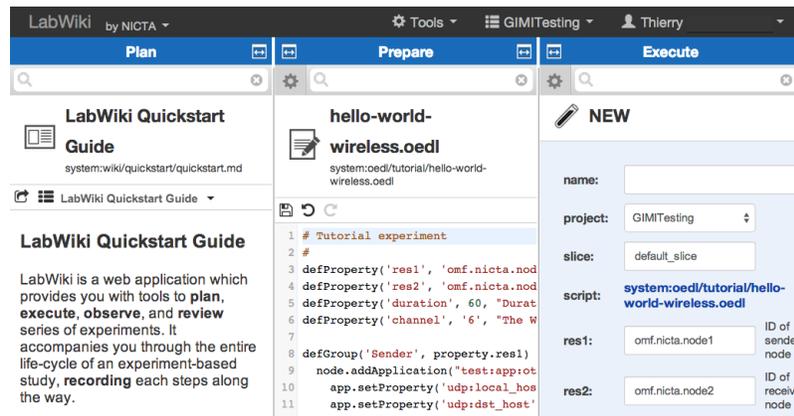
Fig. 3 Main interface of LABWIKI.

LABWIKI, as shown in Figure 2, is sitting on top of a suite of supporting tools and services, which can be used directly by an experimenter, or more likely by other tools acting on her behalf, specifically the SliceService (Section 8.3) which harmonizes resource provisioning across many different testbeds; OEDL (Section 7), a domain-specific language for describing the orchestration of an experiment; Job-Service (Section 9) for scheduling an experiment; OMF & FRCP (Section 9) for executing and coordinating individual experiment runs (or trials); and OML (Section 7) for collecting and managing measurements during a trial.

In addition LABWIKI can be easily extended through *plugins* to extend it's functionality or adapt it to a new environment. Example plugins described in this chapter are the Topology Editor (Section 8.2), Experiment Executer (Section 9.1 and right panel in Figure 3), the Analysis Widget (Section 10), and the iBook Widget Creator (Section 11.2).

3 LabWiki User Experience

As mentioned in the Introduction, the experimenter interacts with LABWIKI primarily through a web browser. After a standard login process, the user will see (Figure 3) a browser window split into three columns, labeled "Plan", "Prepare", and "Execute". This reflects the basic workflow identified above. Each column comprises a tool & search bar, followed by a widget header, an optional widget toolbar and the widget body. The top tool & search bar allows the user to quickly locate or create resources relevant to the respective activity and choose the desired widget to interact with that resource. LABWIKI itself is a framework with most of the functionality provided by plugins, which in turn provide one or more widgets. For instance, the *wiki* widget for the "Plan" column supports editing of rich text resources.

All widgets are stateless and only provide mechanisms for a user to interact with one or more named resources. These resources may reside on a separate service, such as the JobService (Section 9.2), or are file-like resources, such as a wiki entry, or an image. For these kinds of objects, LABWIKI provides layered, pluggable artifact stores. Current implementations support persistence through the local filesystem, versioned and access-controlled repositories such as Git [2], and via iRODS [16]. The clean separation of stateless widgets and state-full, externally resolvable resources allows for interacting and embedding of these resources outside of LABWIKI as well. For instance, plots of experiment measurements, hosted on JobService (Section 9.2) can be embedded into a wiki page, which then can be published from the *wiki* widget to a third-party blog service. Importantly, the link from the plot in the blog entry to the actual experiment is maintained, including access control mechanisms.

LABWIKI supports multiple user accounts and uses OpenID for authentication. Resources, managed through LABWIKI belong to projects and a user's membership and role in a project are the basis of LABWIKI's authorization mechanism. Information about membership and respective roles are sourced from external services, such as the GENI ClearingHouse. Currently LABWIKI is also facilitating the transfer of delegation and *speaks-for* credentials for the services some of the plug-ins call upon (e.g. SliceService Section 8.3).

4 Experiment Overview

As the main objective of LABWIKI is to support a group of researchers in producing verifiable experiments, we will use a real research experiment as the guide through the reminder of this chapter. This experiment was first designed as part of a research effort on time synchronization in networked sensors, with the results published in [13].

Researchers in many domains, such as human-computer interaction, are increasingly collecting large amounts of data from heterogeneous distributed sensors. Accurately synchronizing these data streams is crucial for meaningful analysis and conclusions. While there are many, well-established techniques for synchronizing clocks in distributed entities [14, 19], they require additional software to be deployed on these entities, or depend on variables which may not be under the experimenter's control (e.g., the offset between a NTP client and a server depends on the network's round-trip delay). The above mentioned research project proposed a different approach based on measurements of the data collection system itself and uses the obtained meta data to synchronize the original data a-posteriori.

The main experiment assumes a scenario where certain events can be measured by more than one sensor and where all sensors then forward these measurements to a common collection server. Figure 4 illustrates the resulting experiment topology. A series of events are generated by a source S and measured by two entities $E1$ and $E2$. The respective measurement samples are sent to the same collection server

C. Time delays may be added at the various *Dij* points. *E*1, *E*2, and *C* add locally sourced timestamps *t* to all samples that they produce and receive, respectively.
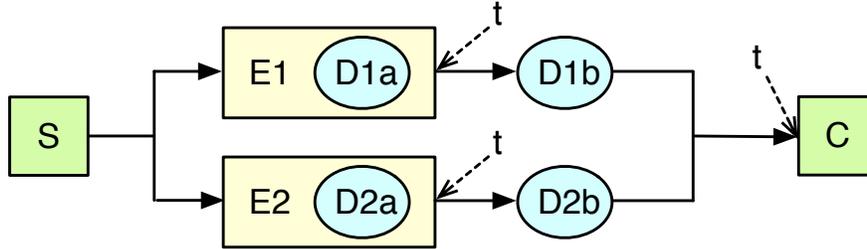
Fig. 4 Topology of the Time Calibration experiment from [13].

5 Experimental Facilities

Major initiatives such as GENI [1] and FIRE [5] have focused on providing distributed, virtual laboratories for transformative, at-scale experiments in network science and services. Designed in response to the Internet ossification issue, these so-called *testbeds* enable a wide variety of experiments in many areas, including clean-slate networking, protocol design, distributed service offerings, social network integration, content management, and in-network service deployment. Many software tools were proposed to allow operators and experimenters to manage, access and control the resources from these testbeds. The models, protocols, and APIs from some of these contributions are currently converging towards *de-facto* standards within the community.

A RSpec[2] defines a set of resources that can be used in an experiments. These resources may be requested from a variety of GENI or FIRE testbeds, such as Exo-GENI, OpenFlow Mesoscale, or Fed4FIRE [20]. There are three different types of RSpecs, *a*) *the Advertisement* which is sent by an Aggregate Manager (AM) to an experimenter to describe its available resources; *b*) *the Request* which is sent by the experimenter to the AM to describe the resources she wants to reserve; and *c*) *the Manifest* which is returned by the AM to describe which resources have been reserved by the experimenter. These RSpecs are exchanged in the previous sequence between the AM and the experimenter. The requested resources will be available to the experimenter after the successful completion of that sequence.

The Aggregate Manager APIs [1] define a common interface for software to provide, request, reserve, and provision resources over different facilities. They are based on a *slice* abstraction, which is a container for all the resources used in a project. Experimenters are associated with slices and use these APIs to interact with

---

[2] http://geni.net/resources/rspec

various entities (e.g., Clearinghouse, Aggregate Manager) in order to discover, reserve, and provision resources. These interactions are mostly performed through third-party interfaces. For example, Omni[3] is a command line tool used to specify and reserve resources from GENI facilities. It allows *stitching*, a technique to connect resources via layer 2 VLans. In contrast, Flack[4] and Jacks[5] are graphical tools, which allow experimenters to reserve resources and specify RSpecs through a visual topology editor. Finally, JFed[6] is a Java-based tool, which allows experimenters to obtain large distributed topologies using resources from both FIRE and GENI testbeds.

The Federated Resource Control Protocol (FRCP) and the OML Measurement Stream Protocol[7] (OMSP) [12, 17] are two protocols to control resources and collect data from them. They are commonly used in both GENI and FIRE facilities. FRCP defines a short set of asynchronous interactions over a *publish-and-subscribe* system, which allows experimenters to configure resources and instruct them to execute given tasks. The OMF and NEPI control tools both implement FRCP [8, 18]. OMSP defines the format and transport of measurement tuples from producers (e.g., a resource) to consumers (e.g., a storage server). It supports various types of measurements, encodings, and the use of metadata. The OML framework [12] provides an OMSP storage server and a C client library to instrument resources. Other client libraries also exist (e.g., OML4R,[8] OML4Py[9] or OML4J[10]).

6 Experiment Design

The first step of an experimental study is the design of the experiment itself. It is driven by research goals, such as testing a hypothesis, measuring performance, or demonstrating capabilities.

There have been many contributions related to experiment design since the seminal work of Fisher [3]. Examples in the area of computer science include [9, 15]. While there are many variations, a good starting point is the identification of the dependent, independent, and confounding variable sets. The *dependent* variables are measured attributes of the studied system, their analysis will provide answers to the study's questions. The *independent* variables would impact the studied system and modify its dependent variables. The third set of *confounding* variables may be un-

---

[3] http://trac.gpolab.bbn.com/gcf/wiki/Omni

[4] http://www.protogeni.net/wiki/Flack

[5] https://www.emulab.net/protogeni/jacks-doc/

[6] http://jfed.iminds.be

[7] http://oml.mytestbed.net/doc/latest/doxygen/omsp.html

[8] https://github.com/mytestbed/oml4r

[9] https://github.com/mytestbed/oml4py

[10] https://github.com/NitLab/oml4j

known or uncontrollable by the experimenter and may have some effect on any of the former variables.

Given these three variable sets, the researcher then devises an experiment plan where usually the *dependent* variables are measured, the *independent* variables are controlled and varied across different repeated trial batches, and the effect of *confounding* variables are mitigated through techniques such as replication or randomization. The choice of controlled values for the *independent* variables and the number of trials and their repetition depends on the objectives of the study.

The LABWIKI workspace has a set of tools to support the experiment design process. The "Plan" column on the left-hand side of its interface (Figure 5) provides a Wiki widget that allows the experimenter to describe and record her design. This design strives to replace her pen-and-paper laboratory notebook. It currently uses the popular Markdown syntax[11], and figures and plots from other widgets can be easily dragged-and-dropped into the write-up.
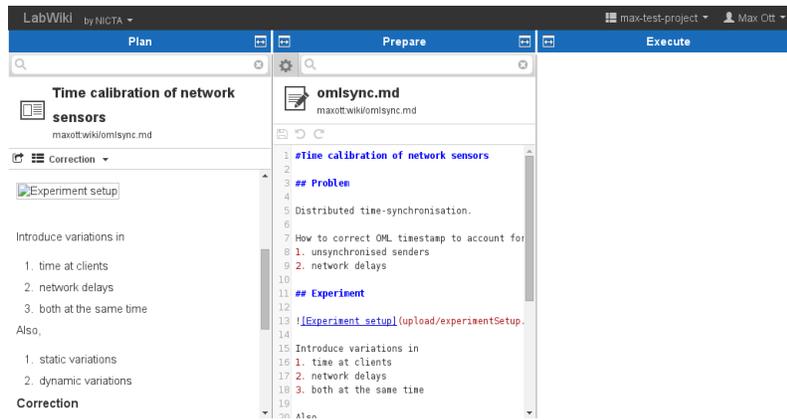
Fig. 5 The experiment design notes are shown in the "Plan" panel. They can be written down in a wiki form in the "Prepare" using Markdown source.

The Design of Our Example Experiment. In the case of our example experiment, we identify the *dependent* variables as the arrival times of a measurement sample at different points in the system. Our *independent* variables consist of configurable clock offsets and network delays, generally referred to as Dij in Figure 4. One potential *confounding* variable would be the varying delays in processing measurement samples inside the sensors.

In this particular realization of the experiment, we chose a classic ICMP ping to the network's broadcast address as the *event* generated by the source S and measured through their respective network interfaces on $E1$ and $E2$. In the first round of trials, the entities are quasi-synchronized and no delay is added on the collection network. This establishes a baseline for future results. We then planned to run a series of

---

[11] http://daringfireball.net/projects/markdown

trials, where various known time offsets are introduced to each entity's clock and on their respective paths on the collection network. To mitigate the potential impacts of *confounding* factors, we decide to run multiple trials for each specific offset configuration and further repeat these multiple trials over different instantiations of our topology. More details about this experiment design and each series of trials are available in the original study [13].

As mentioned above, we primarily use the Wiki widget to describe the design and work plan. Dragging experiment results and other artifacts onto the wiki will allow us to keep track of progress. This will be especially important if an experiment is carried out by a team where different members are pursuing different parts of the work plan.

7 Experiment Description and Instrumentation

Once the design of the experiment is finalized and documented in LABWIKI's Plan panel, the next step is to translate it into a machine-readable description.

7.1 Describing an Experiment

We propose to use the existing OEDL language [18] to describe an experiment. OEDL has been widely used to describe repeatable experiments on both GENI and FIRE facilities. A typical OEDL script is primarily composed of two sections. The first one declares the resources used in the experiment and their initial configurations. For example, an experimenter may declare a given application to be used, along with its available parameters and measurement capabilities; and the specific initial settings for both of them. The second section of the OEDL script defines the orchestration of tasks the resources have to execute throughout an experiment trial. These tasks are grouped into experimenter-defined events, which may be triggered either by timers or experiment-specific conditions. This event-based approach allows complex experiment orchestrations, such as changing the parameters of an application X seconds after the measurement of Y from another application reaches the value Z.

LABWIKI has inspired a third, optional section for a typical OEDL script. It allows an experimenter to define charts to provide quick feedback on the progress or outcome of an experiment trial. The *experiment* widget in LABWIKI uses that to display line, pie, or histogram charts in the respective column with relevant measurement data streams sourced from the JobService. We do want to note, that this is primarily to provide a graphical live feedback on an individual execution of an experiment trial, and will usually not replace a thorough data analysis over the complete result set obtained for an experiment (Section 10).

Listing 1 provides a shortened OEDL script for our example experiment. While the complete OEDL script[12] describes the entire experiment with all required settings as designed in Section 6, this shortened version only shows the experiment for the baseline trials. We will now briefly describe this script and refer the reader to the OEDL Reference document[13] for further details.

```
1  loadOEDL('http://goo.gl/4br2MW')
2  loadOEDL('http://goo.gl/qg8Alo')
3  # From the trace-oml2 package
4  loadOEDL('file:///usr/share/trace-oml2/trace.rb')
5
6  defProperty('entity1','node20','1st_entity_ID')
7  defProperty('entity2','node21','2nd_entity_ID')
8  defProperty('source','node19','Event_source_ID')
9  defProperty('time',180*60,'Trial_duration_[s]')
10
11 defGroup('Entities',prop.entity1,prop.entity2) do |g|
12   # Capture ICMP echo packets
13   g.addApplication('trace') do |app|
14     app.setProperty('filter', 'icmp[icmptype]=icmp-echo')
15     app.setProperty('interface', 'eth1')
16     app.measure('ethernet', :samples => 1)
17   end
18 end
19
20 defGroup('Source',prop.source) do |g|
21   # Broadcast ICMP echo requests every 10s
22   g.addApplication('ping') do |app|
23     app.setProperty('dest_addr', '10.0.0.255')
24     app.setProperty('broadcast', true)
25     app.setProperty('interval', 10)
26     app.setProperty('quiet', true)
27     app.measure('ping', :samples => 1)
28   end
29 end
30
31 defGroup('All',prop.source,
32            prop.entity1,prop.entity2) do |g|
33   # Report time synchronisation every minute
34   g.addApplication('ntpq') do |app|
35     app.setProperty('loop-interval', 60)
36     app.setProperty('quiet', true)
37     app.measure('ntpq', :samples => 1)
38   end
39 end
40
41 onEvent(:ALL_UP_AND_INSTALLED) do
42   group('All').startApplications
43   group('Entities').startApplications
44   group('Source').startApplications
45   after prop.time do
46     allGroups.stopApplications
47     Experiment.done
48   end
49 end
```

Listing 1 Example of an OEDL script

---

[12] http://git.io/clock-delay-impairments.rb

[13] http://omf6.mytestbed.net/OEDLOMF6
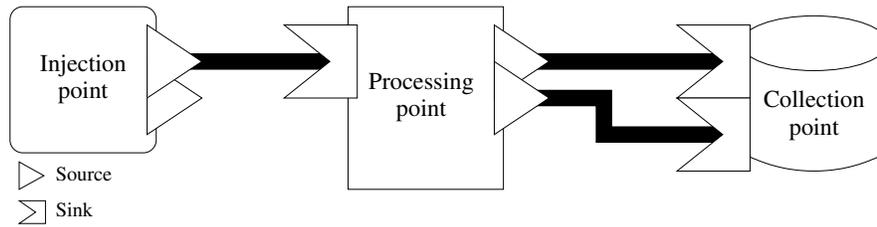
Fig. 6 Applications can be instrumented with OML to *inject* timestamped samples into various measurements streams (MS) which can be *processed* in-line (e.g., aggregated or sub-sampled) before finally being *collected* and written into a storage backend.

Lines 1–4 fetch and load additional OEDL scripts, similar to the *include* statement found in many programming languages. Lines 6–9 define some experiment parameters which may be modified for different trials. Lines 11–18 define a group of resources comprising of the entities $E1$ and $E2$ from Figure 4. An ICMP packet capture application is associated to each resource in that group (l. 13). The parameters and measurements to collect for this application are set using the **setProperty** and **measure** commands, respectively. Lines 20–29 define another group of resource with only the source S from Figure 4. The ICMP ping application is associated to that resource (l. 22), and configured to ping the network's broadcast address (l. 23). Lines 31–39 define a third group which include all the previous resources. A time statistic reporting application is associated to all these resources (l. 34). Finally, Lines 41–49 define the sequence of tasks to perform once all the resources are ready and their associated applications are installed. In this simple case, all the resources first start their time reporting applications. Then the resources within the "Entities" group start their applications. Then the resource in the "Source" group does the same. After a set duration, all resources in all the groups stop their applications, and the trial is finished.

7.2 Instrumenting Resources

LABWIKI leverages the OML measurement framework [12] for measurement collection and storage. OML is based on the concept of *measurement points* (MPs). The schema of an MP defines a tuple of typed metrics meaningfully linked together (e.g., sampled at the same time, or pertaining to the same group). The series of tuples generated by reporting measurements through an MP defines a *measurement stream* (MS). OML defines several entities along the reporting chain that can generate, manipulate, or consume MSs. This is illustrated in Fig. 6.

7.2.1 Instrumentation Process

The instrumentation of a resource consists of enabling applications to act as injection points (Fig. 6). By providing a structured way of defining MPs, OML fosters the reusability and interchangeability of instrumented applications, and simplifies the assembly of subsequent experiments. For example the "wget" and "cURL" applications report similar information about web transfers, and should therefore attempt to reuse the same MPs.

It is therefore important to first identify all the measurements that can be extracted from an application. For example, `ping` not only provides latency information, but also sequence and TTL information for each received packet from any identified host, as well as overall statistics. A rule-of-thumb is that measurements, which are calculated, measured or printed at the same time/line are good candidates to be grouped together into a single MP. For more complex cases, where samples from multiple MPs need to be linked together, OML provides a specific data type for *globally unique identifiers* (GUID). They can be used in a similar way as foreign keys in databases. For example, in the case of the `trace-oml2` application, it was decided to create one MP per protocol encapsulated in a packet (e.g., ethernet or IP). A fresh GUID is generated for each packet, which is then parsed, injecting information about each header in the relevant MP, along with the GUID.

It is also possible to report metadata about the current conditions. Such details as description, unit and precision of the fields of an MP are primordial for the later understanding of the collected measurements.[14] Other information such as the command line invocation, or application version and parameters are also worthy of inclusion as part of this metadata.

7.2.2 Instrumentation Libraries

The most complete OML implementation is the C `liboml2`(1,3).[15] It provides an API, which can be used to define MPs within the source code of an application, and mechanisms to process the injected samples at the source. The `oml2-scaffold`(1) tool can be used to generate most of the boilerplate instrumentation code, along with the supporting OEDL description [12, App. A]. An example application written from scratch to report network packets is `trace-oml2`,[16] used in Listing 1 (l. 13).

The Ruby and Python bindings (OML4R & OML4Py) are particularly useful for writing wrappers for applications for which the source code is unavailable. Wrappers work by parsing the standard output of an application, and extracting the de-

---

[14] Base SI units should be preferred whenever possible.

[15] Manpages for OML system components can be found at `http://oml.mytestbed.net/doc`.

[16] `http://git.mytestbed.net/?p=oml-apps.git;a=blob;f=trace/trace.c`

sired metrics to report. An example is the `ping-oml2` wrapper[17], using OML4R[18], used in Listing 1 (l. 22).

7.3 The Prepare Panel

Our LABWIKI workspace has a "Prepare" panel at the center of its interface (center widget in Figure 3), which provides a simple code editor widget. The experimenter may use this widget to edit an OEDL script, as described previously. All OEDL scripts created within this editor widget are versioned and saved within LABWIKI's artefact store with group-based access control. While this widget is a convenient tool for users to edit their OEDL scripts, the may use alternate means to do so, as well. For example, they may edit their scripts in other editors and then cut & paste it into the "Prepare" panel's code editor, as illustrated in Fig. 7. Alternatively, they may directly use a git repository, and push it into LABWIKI's artefact store.
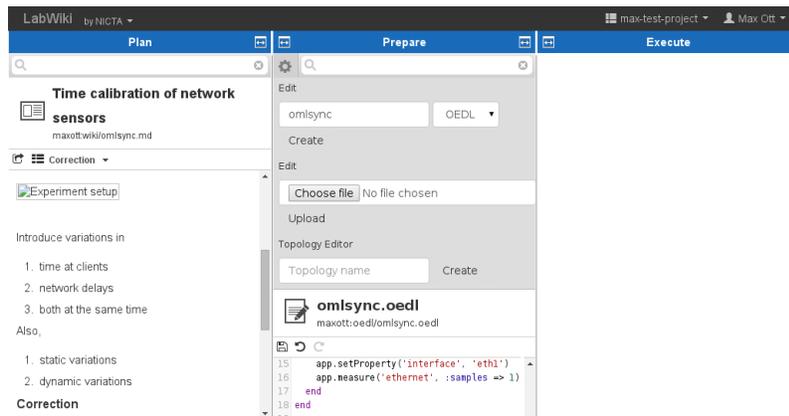
Fig. 7 The "Prepare" panel can be used to upload and edit the various files and scripts needed to describe the experiment (e.g., Markdown or OEDL).

---

[17] We generally use `APPNAME-oml2` as the binary's name of OML-instrumented versions of upstream `APPNAME` utilities; the OEDL application description however only uses APPNAME for conciseness.

[18] `http://git.io/oml4r-ping-oml2`

8 Resource Selection and Provisioning

While the previous section dealt with describing the entire experiment and its resource needs, it did not consider where these resources come from. For instance, the OEDL script in Listing 1 refers to resources, such as virtual machines (nodes) and applications (trace, ping). It is the former kind of resources, which we assume will be provided by testbeds and the programmable networks, as offered by GENI, connecting them.

More specifically, an experimenter needs to first define a topology of nodes, their interconnecting networks, and their specific characteristics. For our example experiment, at least four (virtual) machines and four links are required to create the topology in Figure 4. We note, that should the experiment be extended (e.g., by adding new entities E3 and E4), additional resources have to be reserved. Alternatively, the experimenter may reserve a larger topology and run different trials on a subset of the reserved resources.

8.1 Process Overview

Specification. The very first step is the specification of resources that are required for an experiment. Section 5 described several approaches for resource description and tools to create them. The most common specification is the XML-based GENI RSpec.[19]

Reservation. Once specified, a set of resources needs to be reserved. This requires a negotiation phase between the provisioning tools and the corresponding services on the testbed side. A negative outcome of this negotiation means that the requested resources can currently not be provisioned, or the requestor does not have sufficient privileges, or has exceeded her quota. For example, a VM is requested but all the physical machines' resources have been already allocated to other experimenters. In such a case, the experimenter either waits or hopes that the desired resources will become available in the near future or modifies the request for a different set of resources. A positive response means that the provisioning process will move on to the next step.

Provisioning. After specific resources have been identified, they need to be provisioned before the experimenter can gain access to them. In the case of a physical machine, this may require a power up. In the case of a VM, a disk image containing the requested operating system needs to be loaded and the VM started up with the appropriate configurations. This may also include the distribution of security credentials to limit access to those resources to the requesting party. As each step may take time and in the case of large requests never fully complete successfully, proper communication between the requesting and providing services need to be main-

---

[19] http://groups.geni.net/geni/wiki/GENIExperimenter/RSpecs

tained as even a subset of successfully provisioned services can already be used for successful experimentation.

Monitoring. Most of the resources provided are virtualized in some form or depend on other services in non-obvious ways. It is therefore important for most experiments to be able to monitor their resources and potentially even the broader context in which they are provided. For instance, CPU and memory allocations to VMs may change over time, or there may be external interference in wireless testbeds. While some of these parameters can be monitored by the experimenter herself, others may need special access and therefore need to be collected by the resource provider with the results forwarded to the experimenter. For instance, the BonFIRE [7] testbed provides monitoring information on the physical server to the VM "owners" for the respective server. An experimenter can either use such infrastructure information after the completion of the experiment during the analysis phase or display this information in real time in LabWiki for actual monitoring.

8.2 Labwiki Topology Editor Plugin

LabWiki provides a topology editor plugin, which supports the experimenter in navigating the above described steps. The plugin provides two widgets, one for the "Prepare" panel to specify the topology, and one for the "Execute" panel to request the provisioning of a defined topology and its ongoing monitoring.

Figure 8 shows a screenshot of the first widget, the graphical topology editor. The widget is split vertically with the graph editor on top and the property panel for the selected resource (dotted outline) at the bottom. Interactive graphical editors are usually easy to learn, but do not scale well to large topologies. Hierarchical grouping with associated visual "collapsing" can mitigate some of these scaling issues. However, larger topologies will not be "hand crafted" but generated by tools, such as BRITE [11]. The topology editor has a text-mode, which allows the experimenter to specify a BRITE model as well as provisioning information for the nodes and links created.

The topology description can either be stored as RSpec or extended GraphJSON.[20] It is this textual representation, which the "Slice" widget is sending to the SliceService when requesting the reservation and provisioning of a specific topology. This widget uses the same graph editor (now read-only) to convey progress by animating the graph elements accordingly. Monitoring information is also overlaid to provide experimenters with quick feedback on the overall topology status.
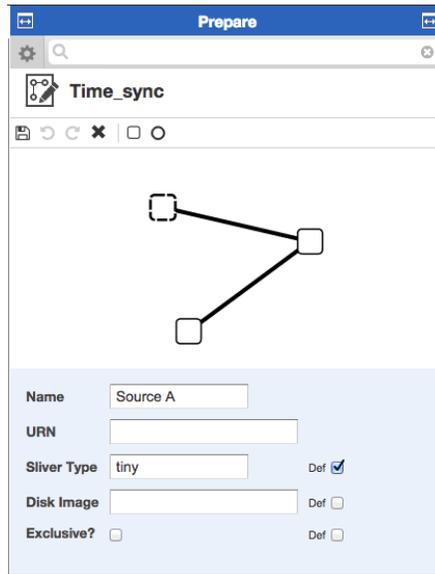
---

[20] https://github.com/GraphAlchemist/GraphJSON

Fig. 8 LABWIKI's topology widget.

8.3 The SliceService

The SliceService provides a REST API for requesting and provisioning of resources
for a testbed federation. It is essentially a proxy service to the SFA APIs of the
ClearingHouse (CH) and AggregateManager (AM). We reluctantly chose this path
as the legacy decisions regarding technology (XML-RPC and client-authenticated
SSL), as well as multiple versions for both API and RSpec put a considerable main-
tenance burden on the upstream tools. Therefore, we designed and implemented a
service, which is based on current best practices for web-based services. We want
to stress, that this is not a value judgement of the SFA APIs but a commonly en-
countered legacy problem. This decision allows us to concentrate our development
as well as debugging efforts regarding testbed interactions on a single service. In
fact, some design decisions for the SliceService have been heavily influenced by
JFed[21], which seems to have similar objectives.

Following the REST philosophy, SliceService defines a distinct set of resources
and provides a consistent set of actions to create, modify, and delete those resources.
It also takes advantage of the recently introduced delegation mechanism based on
credentials. Traditionally, SFA tools were assumed to have access to the requesting
user's private key. However, in this context, the user is authenticated with LABWIKI
which in turn requests SliceService to perform certain actions on behalf of a specific
user. In addition, a specific SliceService instance may serve many different users. To
maintain full transparency on who is operating on whose behalf we need to ensure

---

[21] `http://jfed.iminds.be`

that every request made by SliceService to a CH or AM contains the full delegation chain back to the user. This will allow each CH or AM to decide if it trusts the intermediate services. In turn it increases the security of both LABWIKI and SliceService as user authentication can be delegated to dedicated federation services, such as the CH.

The SliceService also plays a crucial role in the security mechanism of FRCP (Section 5) by providing the newly created resources with proper credentials in a secure manner. However, a detailed description of the overall security mechanism is beyond the scope of this chapter.

9 Running an Experiment Trial

Once an experiment is designed, described, and all necessary resources have been allocated, the next step is to execute an instance (or trial) of that experiment. Running an experiment trial should be effortless for the experimenter, as she will need to repeat this process many times in order to gather sufficient data to derive statistically meaningful conclusions.

9.1 The Execute Panel

Our LABWIKI workspace has an "Execute" panel on the right-hand side of its interface (Figure 3), which hosts an experiment widget. This widget allows an experimenter to configure, start, and observe individual experiment trials, as shown in Fig. 9. To initiate this process, an experimenter intuitively drags & drops the experiment's icon from the previous "Prepare" panel into the "Execute" one. This action triggers LABWIKI to display a list of the experiment properties defined in the respective OEDL script which can now be configured for a specific trial. For example, the experiment design might require that 20 trials should be run with property A set to 1, followed by another 20 trials with A set to 2. Once the experimenter is satisfied with the trial's configuration, she may press the start button at the bottom of the panel, which instructs LABWIKI's experiment widget to post a request for trial execution to an external JobService instance.

9.2 The JobService and Its Scheduler

Our LABWIKI workspace de-couples the frontend interface used to develop and interact with experiment artifacts from the backend processes orchestrating the execution of an experiment trial. The JobService software is the backend entity in charge of supervising this execution. This decoupling enables our tools to cater for
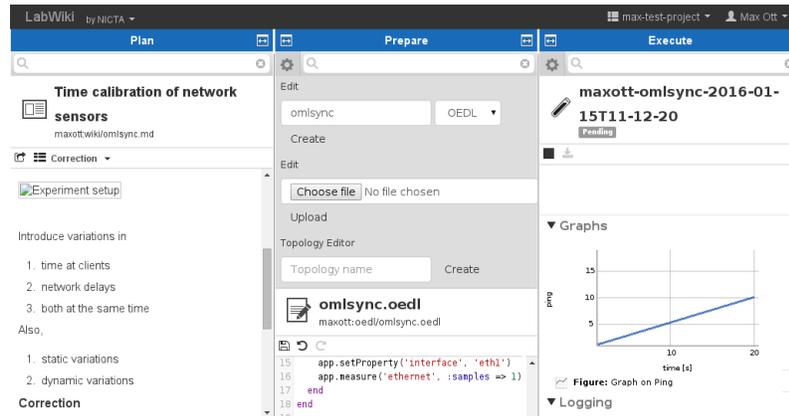
Fig. 9 The "Execute" panel shows details of the currently running experiment, including real-time graphs of the measurements being collected.

a wide range of usage scenarios, such as use of an alternative user frontend, automated trial request (e.g., software can request a given trial to be run at a periodic time), optimization of a shared pool of resources among trial requests from the same project.

The JobService provides a REST API, which allows clients such as a LABWIKI instance to post trial requests (i.e. experiment OEDL scripts, property configuration). Each request is passed to an internal scheduler, which queues it and periodically decides which job to run next. This scheduler function is a plugable module of the JobService, thus a third party deploying its own JobService may define its own scheduling policies. In its simplest form, our default Scheduler is just a plain FIFO queue. However, in an education context it could be a more complex function, which could allow a lecturer to optimize the use of a pool of resources (allocated as in Section 8) between parallel experiment trials submitted by multiple groups of students. The JobService's REST API also allows a client to query for the execution status and other related information about its submitted trials. LABWIKI uses this feature and displays the returned information in its "Execute" panel once the trial execution has started.

9.3 Orchestrating Resources

The JobService uses the existing OMF framework [18], which is available on many GENI facilities to orchestrate experiment trials. More specifically, when the JobService's Scheduler selects a given trial request for execution, it starts an OMF Experiment Controller process (EC). This EC interacts with a Resource Controller (RC) running on each of the involved resources, and have them enact the tasks required in the experiment description. This interaction is done via the FRCP protocol

(Section 5). While the trial is being executed, the JobService constantly monitors the information from the output of the EC process and uses it as part of the status provided back to LABWIKI. While our current JobService uses OMF for its underlying experiment trial execution, its design also permits the use of other alternative frameworks, such as NEPI [8].

9.4 Collecting Measurements

The applications instrumented in Section 7.2 inject measurement streams from measurement points as selected by the experiment description (e.g., Listing 1 ll. 16 or 27). In Fig. 6, the reporting chain is terminated by a collection point. The OML suite [12] provides an implementation of this element in the form of the `oml2-server`(1). It accepts OMSP streams on a configurable TCP port, and stores the measurement tuples into SQL database backends.

In our LABWIKI workspace, the EC instructs the applications on the location of the collection points to report their MSs to. One database is created per experiment, and a table is created for each MP (regardless of how many clients report into this MP). The `oml2-server` currently supports SQLite3 and PostgreSQL databases, and there are plans to extend this to semantic and NoSQL databases.

For very large deployments, the OML collection can be scaled by running multiple `oml2-servers` behind a TCP load balancer such as HAProxy.[22] Instrumented applications carry all the necessary state information in the initial connection for any server to create the tables and store the reported tuples. A PostgreSQL cluster can then be used as a backend to store data into a single centralised logical location where analysis tools can access data both in real time and retroactively.

10 Result Analysis over Multiple Trials

As mentioned in Section 6, it is often necessary to execute several trials of a given experiment to gather sufficient data for a meaningful analysis. LABWIKI facilitates these replicated trials, and provides two options for the experimenter to use the produced data.

First, the "experiment" widget provides an "Export" button once the respective trial has terminated. Depending on the LABWIKI configuration, data can be exported either as a self-contained database file (SQLite3 or PostgreSQL dumps), as a compressed (e.g. ZIP) archive of comma separated (CSV) formatted files, or pushed into an existing iRODS data store. LABWIKI's plugin-based design allows other third-party export widgets to be provisioned as requested. The experimenter may

---

[22] `http://www.haproxy.org/`
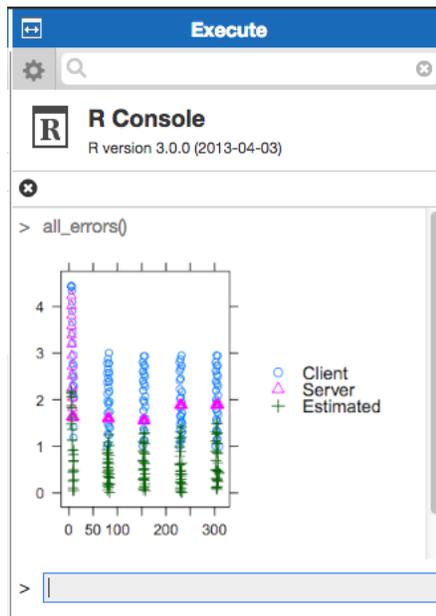
Fig. 10 LABWIKI's analysis widget.

then download the produced database and import it into her preferred data analysis software.

LABWIKI provides another alternative to interact with the produced data through another widget, which interfaces with a separate R server. This widget allows the experimenter to nominate an R script to be submitted to the R server, which executes the script's instructions and returns any outputs (text or graphic results) to the widget for display. Figure 10 shows a screenshot of this widget being used to analyze data from our example experiments.

In many cases, the result analysis of multiple trials will provide new insights into the subject of the study. The experimenter may then reflect on these conclusions and decide to either run further trials with an updated experiment design, description, used resources, and/or analysis as required. This step effectively closes the experiment workflow process as described in Section 1.

11 Store and Publish

As an experiment-based study progresses, it is essential to ensure that all generated artifacts, such as documents, data, and analysis, are permanently stored for continued and future access. Furthermore, once a study finishes, the experimenter should be able to easily share her findings with the community.

11.1 Storing and Sharing with LABWIKI

As mentioned previously, LABWIKI supports different types of artifact stores. This allows different operators to deploy their own LABWIKI instance, to select the storage solution that best suits their requirements. For example, in an evaluation or small closed institution context, a plain directory hierarchy based on the underlying file system may be enough. In contrast, for a large community spanning multiple organizations (e.g. GENI), a solution offering fine- grain access control, high service availability, and version control may be more appropriate. A list of the supported artifact stores are available in our source code repository, together with LABWIKI's "Repository" API, which could be used to add interfaces to other storage alternatives.[23]

To faciliate the sharing of generated documents, LABWIKI's "wiki" plugin has a pluggable export component, which allows the publication of its documents to an external Content Management System (CMS). This feature is accessed through an "Export" action displayed at the top of LABWIKI's "Plan" panel. There is currently only one export plugin available, which allows the sharing of documents to a Respond CMS system.[24] The source code for this component along with documentation on how to implement other ones are available in our repository.[25]

11.2 Publish as a Practical Lab for a Course

Many recent contributions discussed the benefits of using interactive online materials and eBooks in teaching Computer Science (CS) courses [6, 21, 22]. However, the creation of interactive elements (e.g., a practical lab) within an eBook platform currently requires considerable programming skills as well as familiarity with other mechanisms, such as student authentication. LABWIKI provides a simple yet effective solution to generate such an interactive element from an existing experiment. Using this solution, a CS lecturer could develop an experiment in LABWIKI, which would involve resources on distributed testbed facilities and illustrate some aspect of a course. Once finalized, this experiment could then be "embedded" within an eBook. The students using this eBook would then be able to execute a real experiment trial through the eBook interface, i.e., a trial actually using real testbed resources.

Lecturer Side. Once the lecturer is satisfied with her experiment, she needs to generate an eBook widget [10] to act as the interface to that experiment. Such a widget is a self-contained HTML5/Javascript wrapper, which can be embedded in an ePub3 or Apple iBook document. The "Execute" panel of LABWIKI provides a "create widget" action as shown in Figure 11. The lecturer triggers this action and provides

---

[23] http://git.io/omfweb

[24] http://respondcms.com/

[25] http://git.io/labwiki-plan

some configuration parameters for the widget to be generated (e.g., a name, the display size of the widget in pixel, the set of allocated resources). The "create widget" process then generates a fully configured widget, which will be downloaded to the lecturer's machine. She can then include this HTML5/Javascript widget into her eBook using third-party authoring tools.

Fig. 11 Creating an eBook Widget.

Student Side. A student may then download the eBook and open the page with the forementioned widget. Once triggered inside the eBook, the widget will switch to a full-screen web container connected to the remote LABWIKI workspace. From then on, the student has access to a subset of the previously mentioned LABWIKI features, namely the experiment trial execution (Section 9), the result analysis (Section 10), and the publishing (Section 11.1) features. Thus, she is not allowed to modify the experiment description or the set of resources to use. Once the experiment trial finishes, she may perform any result analysis requested by the lecturer as part of the practical lab, and submit the answers as a LABWIKI generated document (Section 9.1).

12 Conclusion

This chapter described a methodology based on well established principles to plan, prepare and execute experiments. We proposed and described a family of tools, the LABWIKI workspace, to support an experimenter's workflow based on that methodology. LABWIKI enables repeatable experiment-based research in Computer Networking, Distributed Systems, and to certain extends Computer Science in general. We showed how this set of tools leverages large-scale Future Internet initiatives, such as GENI and FIRE, and de-facto protocol and model standards, which emerged from these initiatives. It provides services and mechanisms for each step

of an experiment-based study, while automatically capturing the necessary information to allow peers to repeat, inspect, validate and modify prior experiments. Finally, the LABWIKI workspace also provides tools for sharing all generated artifacts (e.g. documents, data, analyses) with the community. For educators, a seamless mechanism to turn an experiment in an interactive practical lab for teaching Computer Science is provided.

Acknowledgments

References

[1] M. Berman et al. 'GENI: A federated testbed for innovative network experiments'. In: *Computer Networks* 61.0 (2014). Special issue on Future Internet Testbeds – Part I, pp. 5 –23. ISSN: 1389-1286. DOI: `10.1016/j.bjp.2013.12.037`.

[2] S. Chacon. *Pro Git*. 1st ed. Aug. 2009. ISBN: 1430218339.

[3] R. A. Fisher. 'The Design of Experiments'. In: (1937).

[4] A. Gabriel and R. Capone. 'Executable Paper Grand Challenge Workshop'. In: *Procedia Computer Science* 4.0 (2011). Proceedings of the International Conference on Computational Science, {ICCS} 2011, pp. 577 –578. ISSN: 1877-0509. DOI: `10.1016/j.procs.2011.04.060`.

[5] A. Gavras et al. 'Future Internet Research and Experimentation: The FIRE Initiative'. In: *SIGCOMM Comput. Commun. Rev.* 37.3 (July 2007), pp. 89–92. ISSN: 0146-4833. DOI: `10.1145/1273445.1273460`.

[6] G. Jourjon, T. Rakotoarivelo and M. Ott. 'From Learning to Researching, Ease the Shift through Testbeds'. In: *Proceedings of TridentCom*. May 2010.

[7] K. Kavoussanakis et al. 'BonFIRE: The Clouds and Services Testbed'. In: *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*. Vol. 2. 2013, pp. 321–326. DOI: `10.1109/CloudCom.2013.156`.

[8] Y.-H. Kim et al. 'Enabling iterative development and reproducible evaluation of network protocols'. In: *Computer Networks* 63 (2014), pp. 238–250.

[9] B. Krishnamurthy, W. Willinger, P. Gill and M. Arlitt. 'A Socratic method for validation of measurement based networking research'. In: *Computer Communications* 34 (Jan. 2011), pp. 43–53.

[10] M. Langer. *iBooks Author: Publishing Your First eBook*. Flying M Production Inc., 2012.

[11] A. Medina, A. Lakhina, I. Matta and J. Byers. 'BRITE: An Approach to Universal Topology Generation'. In: *Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems, MASCOTS'01*. 2001.

[12] O. Mehani, G. Jourjon, T. Rakotoarivelo and M. Ott. 'An Instrumentation Framework for the Critical Task of Measurement Collection in the Future Internet'. In: *Computer Networks* 63 (Apr. 2014). ISSN: 1389-1286. DOI: `10.1016/j.bjp.2014.01.007`.

[13] O. Mehani, R. Taib and B. Itzstein. 'Time Calibration in Experiments with Networked Sensors'. In: *LCN 2014*. Edmonton, Canada, Sept. 2014. ISBN: 978-1-4799-3780-6/14.

[14] D. Mills, J. Martin, J. Burbank and W. Kasch. *Network Time Protocol Version 4: Protocol and Algorithms Specification*. RFC 5905. June 2010.

[15] V. Paxson. 'Strategies for Sound Internet Measurement'. In: *The Internet Measurement Conference (IMC)*. Oct. 2004.

[16] A. Rajasekar et al. 'iRODS Primer: Integrated Rule-Oriented Data System'. In: *Synthesis Lectures on Information Concepts, Retrieval, and Services* 2.1 (Jan. 2010). DOI: `10.2200/s00233ed1v01y200912icr012`.

[17] T. Rakotoarivelo, G. Jourjon and M. Ott. 'Designing and Orchestrating Reproducible Experiments on Federated Networking Testbeds'. In: *Computer Networks* 63 (Apr. 2014). DOI: `http://dx.doi.org/10.1016/j.bjp.2013.12.033`.

[18] T. Rakotoarivelo, M. Ott, G. Jourjon and I. Seskar. 'OMF: A Control and Management Framework for Networking Testbeds'. In: *SIGOPS Oper. Syst. Rev.* 43.4 (Jan. 2010), pp. 54–59. ISSN: 0163-5980. DOI: `10.1145/1713254.1713267`.

[19] K. Römer, P. Blum and L. Meier. 'Time Synchronization and Calibration in Wireless Sensor Networks'. In: *Handbook of Sensor Networks*. Sept. 2005. Chap. 7. DOI: `10.1002/047174414x.ch7`.

[20] W. Vandenberghe et al. 'Architecture for the Heterogeneous Federation of Future Internet Experimentation Facilities'. In: *Future Network and Mobile Summit, 2013*. July 2013. ISBN: 978-1-905824-37-3.

[21] A. Wright. 'Tablets Over Textbooks?' In: *Communication of the ACM* 55.3 (2012).

[22] Y. Zhuang et al. 'Taking a Walk on the Wild Side: Teaching Cloud Computing on Distributed Research Testbeds'. In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. SIGCSE '14. Atlanta, Georgia, USA: ACM, 2014, pp. 535–540. ISBN: 978-1-4503-2605-6. DOI: `10.1145/2538862.2538931`.