

# Experimentation Made Easy with the AMazing Panel

João Martins  
Instituto Telecomunicações  
Universidade de Aveiro  
Aveiro, Portugal  
jmartins@av.it.pt

João Paulo Barraca  
Instituto Telecomunicações  
Universidade de Aveiro  
Aveiro, Portugal  
jpbarraca@ua.pt

Diogo Gomes  
Instituto Telecomunicações  
Universidade de Aveiro  
Aveiro, Portugal  
dgomes@ua.pt

Rui L. Aguiar  
Instituto Telecomunicações  
Universidade de Aveiro  
Aveiro, Portugal  
ruilaa@ua.pt

## ABSTRACT

Experimental testbeds for evaluating solutions in computer networks, are today required as a complement to simulation and emulation. As these testbeds become larger, and accessible to a broader universe of the research community, dedicated management tools become mandatory. These tools ease the complex management of the testbed specific resources, while providing an environment for researchers to define their experiments with large flexibility. While there are currently several management tools, the research community is still lacking tools that smooth the experimentation workflow. These were key aspects that we considered when developing the management infrastructure for our wireless testbed[4] (AMazing). We developed a experimentation support framework supported by an attractive GUI, automation and scripting capabilities, as well as experiment versioning and integrated result gathering and analysis.

## Categories and Subject Descriptors

C.2 [Computer Communication Networks]: Network Operations;  
D.2 [Software Engineering]: Management; H.4 [Information Systems Applications]: Miscellaneous

## General Terms

Experimentation, Management, Measurement

## Keywords

network, testbed, experimentation, web, wireless, omf

## 1. INTRODUCTION

The Internet, and networks in general, are in many ways part of our daily life. As their usage increases, their faults and limitations also become more apparent, motivating network researchers to evaluate and develop new solutions anticipating future scenarios and overcoming the challenges identified. These challenges

vary across a wide range of networking areas, from traffic optimization in telecom operators, to scalability in wide area networks and broadband multimedia streaming, or even to the contextualization of communications in order to optimize usage. For researchers to create reliable network solutions, able to be included in our everyday life, these solutions must be carefully designed, tested, and then put into a cycle of successive refinement.

Simulations are an inexpensive and controllable method of evaluating solutions, especially appropriate for initial approximations to new technologies. However, most simulations rely on simple network models, use mathematical based propagation models, often neglect end terminal complexities, and sometimes considering unrealistic interaction assumptions. Therefore they frequently don't accurately represent the complexities of a highly dynamic environment, such as those resulting of practical usage of IEEE 802.11[9] and other wireless technologies.

Under some conditions, network emulation is an alternative approach to simulation, since it enables instantiations of proposed solutions closer to the real world. Emulation consists of the reproduction of simple scenarios, with a reduced number of physical entities, being some of those entities actual devices, and others simulators. This is a valid approach but is limited in the reproduction of environments closer to real life. Emulating aspects such as radio coverage and link quality is possible, but can result in misleading results (e.g., using MAC filtering to simulate reduced radio range), and these effects can also be observed at higher layers of the protocol stack [6]. Consequently, experimentation using real hardware, and preferentially real users, is the most trustworthy method of validating solutions, under real-world conditions, or close to real-world conditions. Under this assumption, several testbeds have been deployed worldwide [7].

This experimental approach comes with some limitations attached. Testbeds are frequently created within the scope of a funding grant, and around a project. Its construction involves considerable deployment and operation costs, supported by that grant. Once the project is complete, if no other funding appears, and due to the inherent operational costs, the testbed falls into disuse, resulting in a waste of resources. Furthermore, testbed usage is hard for theoretical researchers. They typically lack easy instrumentation and maintainability capabilities, requiring users to have a deep understanding of the testbed software and hardware. More importantly, they frequently lack repeatability capabilities as they lack user interfaces and methods for the coordination of multiple experimental runs. Albeit testbed management systems do exist, they are centered on the equipment management issues and not on the exper-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

WiNTECH'12, August 22, 2012, Istanbul, Turkey.  
Copyright 2012 ACM 978-1-4503-1527-2/12/08 ...\$15.00.

experimenter interests in managing a set of tools. This work focus in the last aspect: providing an interface able to evaluate solutions in a repeatable manner, making it possible to easily identify the impact of changes to the solutions under evaluation in wireless based testbeds.

Currently, there are some testbed management systems available. The core of most existent solutions can be reduced to a couple of shell scripts to operate the testbed resources, without the proper support for experiment automation. Running experiments is thus possible and flexible, but cumbersome, labor intensive, and error prone. The existent solutions lack proper graphical interfaces facilitating user interaction, while centralizing the whole process of configuring experiments, reserving resources, scheduling, and collecting its results. Furthermore, collaboration of multiple parties is vital in the current research environment. Experiment results must be shared with a closed number of peers in order to allow a better analysis and a more solid validation inside research terms.

This paper will present the solution developed for addressing this problem in the AMaZING testbed [4], centering the experimenter workflow as the central target of the testbed management system. Section 2 contains an overview of current tools developed for the purpose of managing testbeds, as well as the current user interfaces supporting them. Section 3 discusses with greater detail one of the most common testbed management systems (OMF), which served as basis for the AMaZING testbed management system. Section 4 presents our management system and Section 5 describes how a typical experiment would be developed. Our closing arguments and future work is presented in Section 6.

## 2. RELATED WORK

As the complexity of both networks and software grows, sophisticated tools must be developed to support rigorous experimentation in current testbeds. Real-world experimentation provides reliable results with the exchange of higher complexity on deployment and management tasks. Next we will describe the available tools that assist the research community to conduct these experiments in the different testbed deployments.

The ORBIT Testbed made a significant contribution towards automated experimentation. It introduced a framework to manage resources, and instrumentation capabilities on testbed resources [17]. Tasks such as collecting measurements or environment properties become more automated, so the user can focus on the design of scenarios and refining solutions. ORBIT later turned this package into a generic framework for the management of testbeds, namely the cOntrol and Management Framework (OMF) [16], which is already used in many deployments across the world. They created a Redmine based portal, to support resource reservation and measurement analysis[12]. Similar initiatives[15], replicating OMFs features, created restricted, yet similar frameworks. MiNT focused in extending the experiment capability of the physical infrastructure [8] by combining the NS-2 simulator [2] with the hardware nodes. PlanetLab, originally a wired testbed, delivers an overlay network, bringing seamless integration of different services. It brings several resources, distributed across many institutions, together to build a planetary testbed[1], and recently has focused towards federation mechanisms and the addition of wireless technologies[3]. Tools such as namely Stork or Gush assist users to deploy software in nodes, with CoMon easing troubleshooting and monitoring in the testbed. ProtoGENI also has similar objectives having tools such as Flack and INSTTools that streamlines resource allocation and measurements collection. Emulab testbed provides an environment that promotes experimenters methodology through his multi-user workbench[18]. More recently, the NEPI[13] Framework was in-

troduced to unify all these different environments, mixing the use of network simulators, OMF-enabled testbeds, and even PlanetLab. The DES TestBed Management System (DES-TBMS) [5] are composed of a subset of components responsible for different tasks in the experimentation process. Each application targets monitoring, scheduling and monitoring of the experiments. A XML file contains network configurations, log files to collect and actions to make during the experiment. This script is evaluated and actions taken in the Testbed, whereas log files are collected and delivered in a database for later visualization, which also represents another component. An additional tool also displays the overall testbed state. ASSERT[14] enhanced their testbed usability by developing a set of applications that simplified experiments creation on their testbed.

While some of tools mentioned here target mostly non-wireless testbeds, they are referred as part of current experimenter tools. These tools aim to solve management of a large-scale geographically distributed testbed, while suggesting important features to have in a testbed deployment.

All of these management systems share similar problems for wireless testbeds: lack of integrated environment, steep learning curve due to custom description languages and methodologies, and lack of reproducibility and optimization driven functionalities. Measurement analysis is reduced to manually editing log files; software deployment of target images and execution of the experiments are usually done manually, which voids most attempts of reproducing the timing of most events. No testbed easily allows running the same experiment multiple times, with the possibility of varying a small aspect and comparing results. While these issues are much desired by the research community, attempts to develop a functional integrated environment are recent, and mostly incomplete in critical ways.

## 3. OMF: CONTROL AND MANAGEMENT FRAMEWORK

The cOntrol Management Framework (OMF) [16] framework is a set of popular tools allowing control, management and measurement collection of testbed resources. Testbed administrators obtain a system providing effective management of all resources, while users can instrument, run experiments, and collect all results in a centralized way. It provides unique capabilities in terms of controllability and operation in a testbed. Currently, it is one of the best tools providing management functionalities for experimental testbeds, while being able to be deployed at a wide range of facilities. It considers the existence of a set of experiment nodes, having multiple radio technologies and local storage, and a set of support servers which provide storage for results and disk images to be loaded by each experiment. These servers also host control software providing an interface to users, and manage the execution of the experiments.

The OMF provides a command line interface (CLI) for users to create their experiments and a simple web GUI to observe its status running. The greatest advantage is the OMF Experiment Definition Language (OEDL) which allows the description of experiments with great detail, similar to the approach followed by simulation tools such as NS-3, and OMNET++. Using OEDL, a script can be created describing all aspects of nodes, as well as applications, events and data collection processes. The major hurdle is the learning curve associated to learning an application specific language (common to most languages). Experiments are defined in a domain language based in Ruby, which can and must be validated prior to execution. Some applications are directly integrated with

OEDL, allowing the definition of application specific parameters in the OEDL script. More advanced experiments, demanding custom applications or complex measurement collection, require deploying the hooks interfacing OEDL and the application.

The OMF Measurement Library [11] was developed to target these issues associated with collecting application-specific metrics, and aiming to unify the way results are collected. It consists of an infrastructure whose clients (applications) inject metrics to a central server that is responsible for its aggregation and deliver the results at the end of the experiment, by means of a SQLite database. Data can be queried offline and even visualized using tools such as MatLab. The OML bundles a traffic generator/receiver, and proper integration with the iperf (and many other tools) for unfamiliar users of these kind of tools. This library was created taking into consideration the usability of results collection, also providing an easy method to instrument third-party applications.

The network community, needs regarding integrated management system integration, is growing. Solutions are required to provide the proper abstraction of the underlying management tools. So far, several advancements were made into the development of tools that largely improve tasks such as monitoring, software deployment and measurement collection. We will tackle these issues to provide an attractive interface that promotes automation and repeatability of such processes, allowing one to create experiments effortlessly: our focus is on the experimenter experience, and we rely on OMF for the testbed management tasks.

#### 4. THE AMazING MANAGEMENT SYSTEM

The AMazING testbed [4] is composed by twenty-four nodes located in the rooftop of Instituto de Telecomunicações - Aveiro. A customized version of OMF is used to provide testbed management functions, automating otherwise tedious but required tasks such as network configuration, software deployment, and experiment execution. In addition we further developed a management overlay centered on the experimenters' workflow, allowing users to create and schedule experiments, as well as analyze the results produced. This management overlay constitutes an extension to OMF that exposes in a user friendly way its functionality for operation in a testbed, and will be detailed in the following sections.

In a standard OMF deployment users must submit their experiment definition, written in OEDL, through the Experiment Controller (EC) loading the system images provided by the Aggregate Manager (AG). Each user must have an EC instance deployed on their machine to be able to run an experiment. Our intent was to extend this deployment (Figure 1) with an additional component, called the AMazING Panel<sup>1</sup>, that aggregates OMF sub components and functionality under a common, web oriented, multi-user management application, and streamlines testbed management, hiding most of the complexity associated to running experiments. It presents the testbed user with an interface oriented to the experiments, and not to the testbed administration.

The Panel development was based on the Ruby on Rails framework, and consists of an application that allows users to interact with the testbed through a graphical interface, while at the same time allowing administrators to manage users and some aspects of system operation. All OMF functionality is available through the interface created. Therefore, we will focus our description in the components and functionalities added or enhanced.

<sup>1</sup>Documentation and Code freely available at <http://helios.av.it.pt/projects/amazing-panel>, testbed interface available at <http://amazing.atnog.av.it.pt>

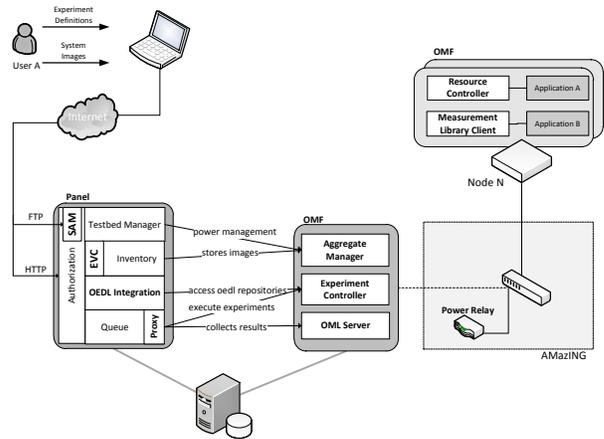


Figure 1: OMF architecture extensions

- **Testbed Manager** - Accesses the chassis manager, controlling the electrical power provided to each node.
- **Queue** - A FIFO Queue is used as the schedule mechanism for experiments execution. Experiments are executed one at a time. The AMazING usage model is "lease-time", meaning that at a given time the whole testbed is dedicated to a single experiment.
- **Proxy** - Is the component responsible for the testbed operation, by preparing and executing experiments, on behalf of the users. Most importantly, it exposes OMF commands through a service oriented interface.
- **Integrated Experiments Environment (IEE)** - An user interface guides users in the development and execution of their experiments. For new users of the platform, it simplifies the learning process when using an OMF-enabled testbed. An OEDL implementation (which powers experiments drivability) was developed in order to extract information from scripts to deliver a better user experience in the panel.
- **Authorization and System Accounts Manager (SAM)** - Users are divided in different roles and can access the panel using both HTTP and FTP. Besides controlling user accesses to the panel, SAM is in charge of managing the FTP accounts exposed to the FTP Server.
- **Inventory** - Is the main data repository. All the experiment definitions, system images and experiment related data are stored in this module.
- **Experiment Version Control (EVC)** - This module is responsible for supporting experiment versioning and the creation of branches.

In the following subsections, we describe the most relevant components of the Panel in detail and present an overview of the interaction between them.

#### 4.1 Integrated Experiments Environment

The most visible aspect of our system is the Integrated Experiment Environment (IEE), which presents users with a graphical

interface allowing them to collaborate in the realization of experiments.

The architecture of this component is divided into two parts: i) the presentation layer consisting in the UI component management; and ii) the engine, which handles code generation (OEDL) and OMF related data. Web services supporting the script code generation, repository auditing, and user application management were developed. All data exchanged makes use of the JavaScript Object Notation (JSON) format.

This module is thus divided in two main components:

- The Engine responsible for handling all data related service invocations. It keeps the state of active experiments, maintaining information about groups, event timeline, resource properties, and applications deployed to experiments. These applications are software packages (e.g. traffic generators) which are required by users for the purpose of executing a given experiment. The engine fetches the definition of OEDL applications, described in the OMF script repositories to later be made available. When a user refers such application in an experiment, all information about its properties comes from this reference. It also maintains information related to user actions: defined groups, applications, network properties and application properties. JSON data is generated with all this information, to be later consumed by the web services invoked by the process of creating the experiment definition.
- The UI component is responsible for input handling, sending the appropriate data the engine will maintain during the experiment creation. This component is divided between content generators, helpers and templates used to generate the proper content.

An environment was created to support these modules, which defines a custom implementation of the OEDL (later described). The purpose is to enable the management system to more easily audit scripts definitions, and to scan OMF repositories and user experiment definitions for matching application. The experiment definition will be generated according to user actions in the IEE. The code generator will be responsible for taking the data from the IEE Engine, and for delivering the source code of the experiment and application definitions to the OMF components.

## 4.2 Proxy

Interaction with OMF internal components is not directly done by the IEE, but through an intermediate component, the Proxy. Its role is to mediate the interaction between the web application and the testbed. Since OMF does not provide an API for developers to programmatically use the testbed, other than the EC CLI, the Proxy fulfills this role by exposing the missing API. The assets (system images, nodes, experiment definition) will be passed down to this proxy, being the proxy's responsibility to communicate with the regular OMF entities.

The Proxy API follows a Service-oriented Architecture and exposes a service interface supporting the methods which the web application uses. Behind these services, state machines control the preparation, and execution behavior of the experiments as reported by OMF. The current implementation specifically maps the OMF commands to web services, so that all functionality provided by OMF is kept.

The OMF Command Line Interface (OMF CLI) provides commands to load an image into nodes and to execute the experiment

definition script. When the Proxy component prepares an experiment it will invoke the OMF load command for the images required by the experiment. Later, in order to check the experiment state, the web interface can inquire the Proxy with a stat command. When the experiment starts, it executes an experiment script (one or multiple times) using the OMF exec command. A new preparation may be issued according to the experiment overall state, when nodes or system images are modified. In each of these proxy routines, all the generated logs and important files are automatically stored and organized within the experiment repository, and available to users in the interface, or through an FTP account.

## 4.3 OEDL language implementation

OEDL is the way users define an experiment and its flow in a OMF-enabled testbed. OEDL script describes valuable data such as the nodes configuration data, applications deployed and metrics to be collected. Current OEDL engine implements each of the domain language methods affecting the overall testbed/nodes state. For other purposes besides experiment execution, the data described in the experiment/application definitions cannot be obtained, making it difficult to provide statistics or better user experience through the web interface.

To support this integration, we built a custom implementation of OEDL, which is responsible for extracting all data contained in the scripts. Besides experiments, this module also fetches information from the Experiment Controller script repositories, which contains a description of the OMF bundled applications included in our provided baseline system image. An example of such integration is the ability to know the resources required by an experiment prior to its execution, in order to visualize resource occupation for each experiment, and provide better feedback during the design phase of the experiment.

## 4.4 Experiment Version Control (EVC)

Creating an experiment can be a time consuming event. Disk images must be created, OEDL must be built, and everything must be deployed to the selected experiment nodes. Syntax and semantic errors in the OEDL script, as well as bugs in the application under test or simply hardware failures can hinder the entire process. Moreover, when evaluating new solutions, it is useful the creation of multiple versions of the same experiment where only a particular set of variables is modified. But above all, it is most important to keep track of which changes are made between different runs of the same experiment. The EVC role is to hide such concerns from users, maintaining a track of the experiment assets and results.

The Experiment Version Control provides the means for users to manage experiment versions, which can be run multiple times in a batch, and branched to new experiments. It is based on the same philosophy of popular versioning systems such as SVN and GIT, providing the means to create branches, modify experiments and keep versions (through explicit commits) of the configuration and results obtained. Therefore, users can truly evaluate their solutions through controlled modification of experiment conditions, and a unified view provided by this tool.

Instead of integrating existent versioning solutions, excessively complex for our basic needs, we developed a fully functional versioning system, based on a filesystem structure and text files. Internally, the EVC component considers that each experiment is a repository with one or more branches. Each branch will contain a set of experiment assets that are under version control: the script and resource map (both text files). Additionally, a folder contains a set of runs, each containing log files and database results (*SQLite*) which can be retrieved at a later time. Branch information includes

commit logs and the number of successful/failed runs. All this information is stored using YAML format. While avoiding dependencies, the use of such a simple structure, text based logs and text based experiment definitions, and *SQLite* based results, allows users to easily explore data in their own computers, almost independently of the operating system they use.

Defining a scenario involves both a repetitive and iterative process until it is successfully defined with valid results. With branches semantics, users can fork the experiment in order to create different scenarios perspectives. Combining this tool with EVCs results and changes tracking, quick comparisons can be made with completely different experiments versions.

These versioning features applied to experimentation enables a rapid switch to completely different experiment versions through a well-organized version control system. It allows brief analysis to be made with the results obtained, while having runs historically stored. These are all reasons that make EVC one of the most important components of this system.

## 4.5 System Accounts Manager (SAM)

Additionally to the IEE, which users can explore to interface with the AMaZING Panel, another interface using FTP was provided. The FTP interface does not provide means for users to interact with their experiments, but allows them to manage their library and experiment assets. Besides FTP file transfer being faster than using HTTP, it is important that the user gets access to all the experiment assets, and its resources, as well as results in both processed and unprocessed forms. Moreover, disk images sometimes are rather big files and may imply many changes over different experiment versions. Its transfer is facilitated through the use of FTP. Access to each account is directly controlled from the AMaZING Panel application, and related to the membership information of each workspace. The SAM module consists in this bridge between the AMaZING Panel and the actual FTP service.

All information contained in the experiments (e.g. scripts, system images) is exposed through FTP, and users may download it to their computers. As described previously, even data kept under versioning can be analyzed without resorting to specific software. An important design requirement is that both structure and information is mapped into readily available forms. In this case: directories and text files.

## 4.6 Experiments workflow

In order to make use of the testbed, users are expected to follow an experiment setup workflow, obviously related to the underlying the OMF experiment setup workflow. The workflow is enforced through the testbed management system, which requires from the user a set of initial setups, before any test can be conducted on the platform. One of the testbed management system base concepts is the notion of workspaces, which can be shared by multiple users. Each workspace represents a set of experiments related to the same topic, and having the same set of users associated. The workspace metaphor can be useful not only for researchers evaluating the same set of solutions, but also for teachers. Using this approach, teachers can create workspaces with several experiments ready to run. Students can be added to the workspace and either analyses the results obtained by their colleagues, replay experiments or run new experiments. Visibility of experiment definitions, results and the remaining assets is restricted to members of a workspace.

A Library provides storage capabilities, to where users can up-

load their disk images, which later can be loaded to experiment nodes. Upload of disk images can be done through a web interface or through FTP. Each system image being composed by a complete filesystem, containing all the software components required for interaction with an OMF-enabled testbed, plus user software under test (additional kernel modules, third-party applications, software packages, etc.). The experiment definition contains all logic conducting the experiment, including application definitions, network properties, shell commands, etc. Any user can add experiments to the workspace, or replay any of the public experiments present, and use any system image present in the library. Experiments can be constructed and edited, and the IEE provides a graphical interface guiding users through the entire process of defining the experiment, defining the timeline and applications, select nodes, setup data gathering hooks and even analyze results. This task is made simple through the use of the previous described IEE, which facilitates the configuration processes.

## 5. AN EXAMPLE EXPERIMENT

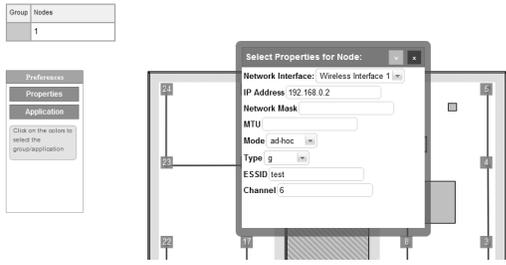
In this section of the paper, we will highlight how the AMaZING testbed management system can improve the process of creating, and executing an experiment, as well as gathering the results obtained. Provisioning of an experiment in the AMaZING testbed system is divided in three steps: configuration of the experiment parameters and software deployed; definition of the execution scenario, and finally configuration of the metrics to be observed and analyzed. After these steps, the experiment is queued for execution, probes are deployed, code is executed, and results are gathered. Finally, users can analyze the data obtained and, if required, branch the current experiment by creating variations of the conditions, which resulting data can then be compared. While the system developed makes use of OMF, it enhances OMF by facilitating the entire experimentation process.

### 5.1 Experiment Setup

In order for OMF to properly operate, it requires the existence of an experiment definition script describing the configuration to be applied to each node, and the set of applications to run at each instant. Running experiments involving integration of applications or execution more complex execution dynamics, will require full understanding of the language used to describe the experiments. In practice, the threshold of defining what is a complex experiment is rapidly reached, and knowledge of the underlying language (OEDL) becomes a requirement for experimenters.

While OEDL presents ultimate flexibility, the learning curve may be too steep for new users; furthermore if the experiment to be executed is simple, this may impose a very large overhead, which discourages usage of highly coordinated systems such as OMF enabled testbeds. Therefore, when using the AMaZING Panel, users are first presented with an interface where, by means of a graphical representation of the testbed and its nodes, they can define which nodes will be used and what configuration is to be applied to each of the nodes. Since in the particular case of the AMaZING testbed not all nodes have direct radio connectivity, having a visual representation of the nodes location greatly helps experiment planning, setup and deployment.

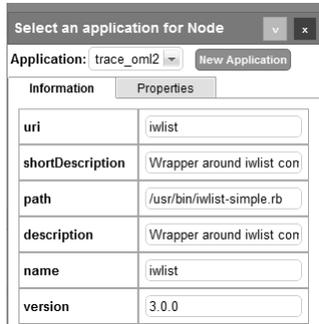
As depicted in Figure 2, the application allows the definition of the network configuration of each node, either individually or in bulk, by aggregating nodes in groups and setting the configuration for all members of the group. Users are also allowed to select or define applications to deploy on nodes, as illustrated in Figure 3. As shown, there is no need for interaction with the OEDL framework, as the application will generate the correct experiment script.



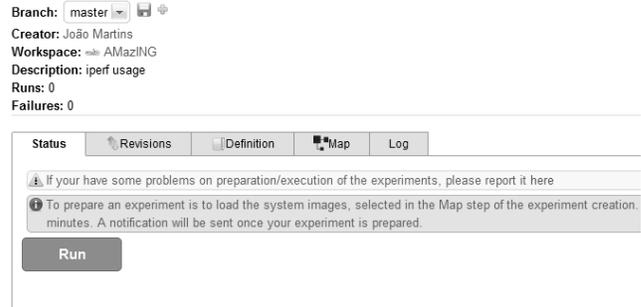
**Figure 2: Configuring experiment parameters.** Left sidebar represents the toolbox. Tables on top indicates nodes/applications configured



**Figure 4: Experiment Timeline, tables illustrate current nodes and applications configured.**



**Figure 3: Per node application definition**



**Figure 5: The experiment management page.**

The next step consists of defining the execution scenario and the timeline of the experiment. Users can visually define their actions in a timescale, allowing applications to start at a specific instant during a certain amount time, or to execute shell commands (e.g. a daemon, load additional kernel modules, poweroff node). As we can see in Figure 4, all actions will be annotated in the timeline which gives an overview of what will happen during your experiment. Ultimately, advanced users can use the Ruby standard library for an even broader range of possible actions.

Once the experiment description script is created, software deployment takes place. Disk images must be configured and deployed on nodes. Construction of these images can be done by users using virtualization tools, and converted to compressed format used by Frisbee [10], which is bundled with OMF. Once the image is created, the user must associate it with the nodes defined in the script that are displayed in the form, therefore finishing the experiment creation step. However note that the predefined images already provide adequate tools for most users, and only require the addition of the software solution under testing.

## 5.2 Execution

After the experiment is setup, it can be queued in the system, issuing either a single or multiple runs. Multiple runs are of particular importance for system analysis, and when evaluating applications over the highly dynamic wireless medium. Users can see the status of the process, and an email is sent once the experiment finishes.

As we can see in Figure 5, this application provides a variety of actions related to experiments, visually distributed through tabs. The user can observe the current status of the experiment, seeing the load progress of images. The revisions tab allows a user to switch to another version of the scenario. The next two tabs is intended for scripts and system images mapping edition. The last one

is saved for logging information when experiment preparation or execution takes place, which is useful for third-party applications debugging or to determine the cause of the failed experiments. Finally, the "Results" tab is reserved for graphics drawing and download, but only appears when an experiment has valid results.

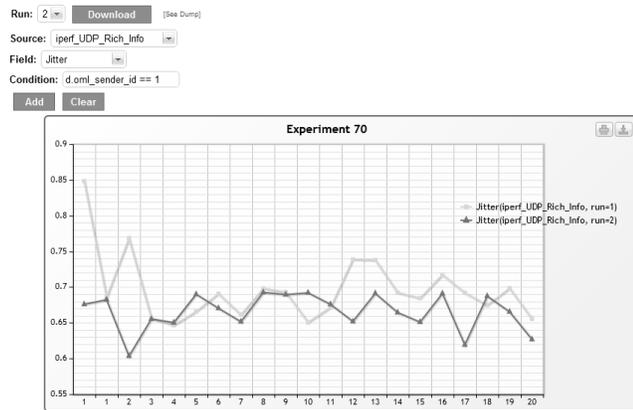
Repeatability in experiments is important for the quality of the results obtained, and for robustness of the solutions under test. The Experiment Version Control (EVC) allows users to isolate different use-cases of the main scenario (e.g. fault tolerance, different traffic profiles, etc.). One can easily switch the whole deployment of images, scripts and results, while preserving the history of the different changes. Accurate results can be obtained by queuing dozens of runs from the same experiment definition, and later download and compare all the results produced. These results are also able to be shared with the research community for later analysis in the platform, or even reference in scientific studies. Peers can also use the platform to re-run the experiment with no additional work at all, or introduce variations and observe modifications in the results.

## 5.3 Result Visualization

Running an experiment can take several minutes or hours. When an experiment reaches its completion, the user navigates to the previously described environment for further analysis of the results obtained. If an offline analysis is preferred, a database with all the results can be obtained and used in tools such as Matlab.

Besides downloading the database with the resulting data, users are able to analyze the results obtained directly in the application. As depicted in Figure 6, we resorted to cross-platform Javascript and CSS technologies, and are able to dynamically create web charts that take into consideration the results obtained by one or several experiment runs. This functionality is very useful when designing the experiment as it allows to rapidly run a variation of an experiment and observe the impact to the results, eventually identifying

flaws in the experiment requiring further adjustment of the timeline, configuration, or the actual solution.



**Figure 6: Graphics visualized include data sets from different runs or branches.**

Without our application, this task involves instrumentation of a high volume of log files, for valid data to be extracted, so that it is later used in tools such MatLab or GNUPlot. The AMazING Panel allows a coarse analysis to be reduced to a few clicks. If custom applications are involved, a small integration with OML is necessary, with the purpose of enabling gathering the relevant metrics. The rest of the work remains delegated to the backend, which provides a dynamically created database with the results. As our testbed tool usage grows, a broader range of OML integrated applications is expected to be available for users to use in their experiments (e.g. VLC for streaming, Mobile IP). Nevertheless, we already support most common tools such as Orbit Traffic Generator and iperf.

We also believe that the educational potential of the AMazING Panel is greatly increased with this functionality. In particular, because it allows the comparison of results obtained in different runs and experiments integrated in the experiment interface, while enhancing repeatability in execution and simplifying complex tasks such as node deployment, and network management.

## 6. CONCLUSION

The design of an experiment involves many steps and passes through an iterative process until a scenario is successfully described. OMF alone only presents means to instrument a testbed, not a complete set of experimenters, and it is not prepared for being used by multiple experimenters through friendly web interfaces. A platform was created to support the AMazING testbed focusing in expanding OMF to become experimenter-friendly. This platform presents a testbed management overlay system where experimentation is facilitated by hiding most of its complexity and proposing that users focus more on the scenarios and results to obtain than on the configuration and deployment of the experiment. It also introduces an environment to create experiments which helps them to adapt better to the whole platform, while allowing a stronger focus on their scientific needs. Furthermore, it enables users to create experiments in a methodological and repeatable way, with version control and batch runs while being supplied with a usable interface to ease most of these tasks.

In the near future we intend to enhance the solution developed, adding support to a richer set of administrative tasks, as well as an

integrated monitoring module for the purpose of monitoring operational metrics (CPU usage, Temperature, etc...) and node status (detect potential problems). Because only a subset of the OEDL language is currently supported, we plan that, in a future version, support for programmatically defined topologies and prototypes can also be defined through the IEE GUI. Finally, enhancements in the mechanisms handling disk images deployed are also envisioned.

We believe that this approach can be replicated in other testbeds and testbed management systems, greatly enhancing the attractiveness of the testbed usage.

## 7. ACKNOWLEDGEMENTS

We would like to thank Rui Ferreira, Carlos Gonçalves, André Rainho and all the users that utilize the AMazING testbed for their feedback.

## 8. REFERENCES

- [1] Planetlab architecture: An overview. <http://www.planetlab.org/files/pdn/PDN-06-031/pdn-06-031.pdf>, February 2006.
- [2] The NS-2 Manual. <http://www.isi.edu/nsnam/ns/doc/>, October 2011.
- [3] The OneLab Project. <http://onelab.eu/>, February 2012.
- [4] J. P. Barraca, D. Gomes, and R. L. Aguiar. AMazING – Advanced Mobile wireless playGrouNd. In *International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities Proceeding, TRIDENTCOM '10*, pages 219–230, 2010.
- [5] B. Blywis, M. Guenes, F. Juraschek, and J. H. Schiller. Trends, advances, and challenges in testbed-based wireless mesh network research. *Mobile Networks and Applications*, 15(3):315–329, 2010.
- [6] R. Chertov, S. Fahmy, and N. B. Shroff. Fidelity of network simulation and emulation: A case study of tcp-targeted denial of service attacks. *ACM Trans. Model. Comput. Simul.*, 19:4:1–4:29, January 2009.
- [7] P. De, A. Raniwala, S. Sharma, and T. Chiueh. Design considerations for a multihop wireless network testbed. *IEEE Communications Magazine*, 43(10):102–109, October 2005.
- [8] P. De, A. Raniwala, S. Sharma, and T. Chiueh. Mint: a miniaturized network testbed for mobile wireless research. In *24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 4 of *INFOCOM '05*, pages 2731 – 2742, march 2005.
- [9] J. Heidemann, N. Bulusu, J. Elson, C. Intanagonwiwat, K. chan Lan, Y. Xu, W. Ye, D. Estrin, and R. Govindan. Effects of detail in wireless network simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 3–11, Phoenix, Arizona, USA, January 2001. USC/Information Sciences Institute, Society for Computer Simulation.
- [10] M. Hibler, L. Stoller, J. Lepreau, R. Ricci, and C. Barb. Fast, scalable disk imaging with frisbee. In *Proc. of the 2003 USENIX Annual Technical Conf.*, pages 283–296, San Antonio, TX, June 2003. USENIX Association.
- [11] J. White, G. Jourjon, T. Rakotoarivelo, and M. Ott. Measurement architectures for network experiments with disconnected mobile nodes. volume 46 of *TRIDENTCOM '10*, pages 350–365. Springer-Verlag, May 2010.
- [12] G. Jourjon, T. Rakotoarivelo, and M. Ott. A portal to support rigorous experimental methodology in networking research.

- In T. Korakis, H. Li, P. Tran-Gia, and H.-S. Park, editors, *Proceedings Testbeds and Research Infrastructures for the Development of Networks and Communities, TRIDENTCOM '11*, volume 90, pages 223–238. Springer Berlin Heidelberg, 2011.
- [13] M. Lacage, M. Ferrari, M. Hansen, T. Turletti, and W. Dabbous. Nepi: using independent simulators, emulators, and testbeds for easy experimentation. *SIGOPS Oper. Syst. Rev.*, 43(4):60–65, Jan. 2010.
- [14] E. Nourbakhsh, R. Burchfield, S. Venkatesan, N. Mittal, and R. Prakash. Enhancing assert: making an accurate testbed friendly. In *Proceedings of the 6th ACM International Workshop on Wireless network testbeds, experimental evaluation and characterization, WiNTECH '11*, pages 3–10, New York, NY, USA, 2011. ACM.
- [15] M. Portoles-Comeras, M. Requena-Esteso, J. Manges-Bafalluy, and M. Cardenete-Suriol. Extreme: combining the ease of management of multi-user experimental facilities and the flexibility of proof of concept testbeds. In *International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities Proceedings, TRIDENTCOM '06*, pages 266–276, 2006.
- [16] T. Rakotoarivelo, M. Ott, G. Jourjon, and I. Seskar. Omf: a control and management framework for networking testbeds. *SIGOPS Oper. Syst. Rev.*, 43(4):54–59, Jan. 2010.
- [17] D. Raychaudhuri, M. Ott, and I. Secker. Orbit radio grid tested for evaluation of next-generation wireless network protocols. In *Proceedings of the First International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, TRIDENTCOM '05*, pages 308–309, Washington, DC, USA, 2005. IEEE Computer Society.
- [18] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, volume 36, pages 255–270, Boston, MA, Dec. 2002. USENIX Association.