Universidade de Aveiro Departamento de
Electrónica, Telecomunicações e Informática,
2011

João Pedro Marçal Lemos Martins

Sistema para gestão remota de redes experimentais

Testbed management systems

Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2011

João Pedro Marçal Lemos Martins

Sistema para gestão remota de redes experimentais

Testbed management systems

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemàtica, realizada sob a orientação científica do Dr. Diogo Nuno Pereira Gomes, Assistente Convidado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Prof. Dr. Rui Luis Andrade Aguiar, Professor Associado com Agregação do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

agradecimentos

Chegado o final do meu percurso académico, é imensurável o número de pessoas a quem gostaria de agradecer. Este espaço não espelha toda a gratidão que sinto por todos os que participaram neste meu percurso.

Em primeiro lugar gostaria de agradecer aos professores Diogo Gomes e Rui Aguiar pela oportunidade de realizar este trabalho e ao João Paulo Barraca, pela sua orientação e colaboração ao longo desta dissertacão. Aproveito também para agradecer às pessoas que compõe o ATNOG pelos conhecimentos transmitidos ao longo da minha formacão. Desse grupo devo destacar, Rui Ferreira e Gonçalo Morais pela disponibilidade e importante contributo.
Aos meus amigos de Aveiro, por me aturarem, mostrando-se sempre disponíveis para ajudar e ouvir.

À Diana pela sua imensa paciência e apoio, animando-me nos momentos mais difíceis.
Em especial aos meus pais e à minha irmã, a quem eu dedico este trabalho. Um grande obrigado por toda a educação, princípios e valores transmitidos, bem como oportunidades e apoio dados ao longo de toda a minha vida. Sem vocês nada disto seria possível.

resumo

A Internet e as redes no seu geral estão cada vez mais presentes no nosso quotidiano. No entanto, o seu crescente uso destaca também algumas das suas limitações, levando os investigadores de redes a criarem novas soluções, numa tentativa de preverem e resolverem problemas que daí possam advir. O desenvolvimento destas soluções implica que sejam realizados testes exaustivos antes que estas se tornem aptas para o uso das massas.

Podem ser adoptadas diversas abordagens para a realização destes testes. Simulações são importantes como uma primeira aproximação, contudo, não conseguem captar a dinâmica e a imprevisibilidade de um ambiente real. A emulação surge como alternativa, tentando colmatar essa falha, no entanto, não é eficaz na reprodução de cenários mais complexos. A solução poderá passar pela criação de infraestruturas dedicadas ao teste destas soluções, denominadas de redes experimentais, permitindo realizar uma avaliação mais profunda e fiável das mesmas. Contudo a sua construção envolve grandes custos e as infraestruturas caem em desuso assim que é alcançado o produto final. É necessário desenvolver maneiras de instrumentalizar estas infraestruturas, de forma a optimizar estes recursos para futuros testes.

Esta dissertação centra-se nos problema evidenciados na criação destas redes de teste, estudando as existentes formas de gestão destas redes experimentais, bem como o seu uso optimizado com vista à criação de experiências. Pretende-se assim desenvolver no âmbito desta tese, uma ferramenta que permita aos utilizadores criarem e usarem facilmente e de forma eficiente uma rede experimental.

abstract

The Internet, as with networks in general, is increasingly present in our everyday lives. However, its use also highlights some of their limitations, leading researchers to create new network solutions in an attempt to anticipate and solve problems that may arise. These solutions require exhaustive tests before they become eligible for production.

Several approaches can be taken to conduct such tests. Simulations are important as first approximations; however, they fail to capture the dynamics and unpredictability of a real environment. Emulations present an alternative, but fail to reproduce more complex scenarios. Experimentation facilities must therefore be deployed, to achieve a higher level of realism. These facilities, also referred as testbeds, involve high-complexity deployment and operation, falling into disuse once they arrive at a final product. Methods of managing these infrastructures must be developed in order to avoid this waste of resources for future tests.

This dissertation focuses on the problems inherent to the management of these facilities, with a focus on their users. Different approaches in network testing must be studied and the different methodologies on experimentation must be collected. An application is developed to address this problem, enabling users to create their experiments and collect results in both an organized and effective manner.

Contents

List of Figures

List of Tables

List of Acronyms

AM	Aggregate Manager
AMazING	Advanced Mobile wIreless Network Playground
AODV	Ad hoc On-Demand Distance Vector Routing
AP	Access Point
APE	Ad-hoc Protocol Evaluation
API	Application Programming Interface
AS	Auditing Service
BSD	Berkeley Software Distribution
BOSH	Bidirectional-streams Over Synchronous HTTP
CDMA	Code Division Multiple Access
CLI	Command-Line Interface
CLR	Common Language Runtime
CM	Chassis Manager
CoTS	Comercial Off-The-Shelf
DES	Distributed Embedded Systems
DES-Cript	DES Experiment Language
DES-Exp	DES Experiment module
DES-Eval	DES Evaluation module
DES-Mon	DES Monitoring module
DES-TBMS	DES TestBed Management System(See DES)
DES-Vis	DES Visualization module
DHCP	Dynamic Host Configuration Protocol
DSL	Domain Specific Language
EC	Experiment Controller
ESSID	Extended Service Set ID
ETag	Entity Tag
EtherBoot	Ethernet Booting
EVC	Experiment Version Control
EXTREME	EXperimental Testbed for Research Enabling Mobility Enhancements
FIFO	First In First Out
FTP	File Transfer Protocol
GIT	GIT
GloMoSim	Global Mobile Information Systems Simulation Library
GML	Graph Modelling Language

GPLv2	GNU General Public License Version v2.0
GPRS	General packet radio service
GS	Grid Service
GT-ITM	Georgia Tech Internet Topology Model
GTNetS	Georgia Tech Network Simulator
GUI	Graphical User Interface
HSDPA	High-Speed Downlink Packet Access
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IDE	Integrated Development Environment
IEE	Integrated Experiments Environment
IETF	Internet Engineering Task Force
INRIA	Institut National de Recherche en Informatique et en Automatique
IOCTL	Input/Output ConTroL
IP	Internet Protocol
IPFIX	Internet Protocol Flow Information Export
IPTV	Internet Protocol Television
IR	Infrared
IT	Instituto de Telecomunicações - Aveiro
JiST	Java in Simulation Time
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
KAIST	Korea Advanced Institute of Science and Technology
MA	Management Authority
MAC	Media Access Control
MIB	Management Information Base
MiNT	Miniaturized Network Testbed
MiNT-m	Miniaturized Network Testbed Mobile
MiNT 2	Miniaturized Network Testbed Mobile 2
MIT	Massachusetts Institute of Technology
MOVIE	Mint cOntrol and Visualization InterfacE
MP	Measurement Point
MVC	Model View Controller
NAM	Network Animator
NBP	Network Boot Program
NED	Network Description Language
NEF	Network Experimentation Frontend
NEPI	Network Experimentation Programming Interface
Net-SNMP	Network Simple Network Management Protocol via Web
NFS	Network File System
NGN	New Generation Networks
NH	Node Handler

NICTA	National Information and Communications Technology Australia Ltd
NitLab	Network Implementation Testbed Laboratory
NM	Node Manager
NS	Network Simulator
NS-2	Network Simulator 2
NS-3	Network Simulator 3
NSE	Network Simulator Emulation
OEDL	OMF Experiment Definition Language
OLSR	Optimized Link State Routing Protocol
OMF	cOntrol and Management Framework
OML	OMF Measurement Library
OMNeT++	Objective Modular Network Testbed in C++
OPNET	OPNET
ORBIT	Open Access Research Testbed for Next-Generation Wireless Networks
OSI	Open Systems Interconnection model
oTcl	Object Tool Command Language
OTG	OMF Traffic Generator
OTR	OMF Traffic Receiver
PARSEC	Parallel Simulation Environment for Complex Systems
PCI	Peripheral Component Interconnect
PLC	PlanetLab Consortium
PLR	Packet Loss Rate
PubSub	Publish-Subscribe
PXE	Pre-eXecution Environment
QoS	Quality of Service
RAM	Random Access Memory
RC	Resource Controller
RFID	Radio-Frequency IDentification
RoR	Ruby on Rails
RPC	Remote Procedure Call
RSSI	Received Signal Strength Indication
RTT	Round Trip Time
RWTH	Rheinisch-Westfaelische Technische Hochschule Aachen University
SA	Slice Authority
SAM	System Accounts Manager
SCS	Slice Creation Service
SFS	Secure File System
SimPy	Simulation in Python
SIP	Session Initiation Protocol
SNMP	Simple Network Management Protocol
SNR	Signal to Noise Ratio
SVN	Subversion
SWANS	Scalable Wireless Ad hoc Network Simulator

TCP	Transmission Control Protocol
TTL	Time To Leave
TFTP	Trivial File Transfer Protocol
UV	Ultraviolet
UCSB	University of California, Santa Barbara
UCSB Meshnet	University of California Santa Barbara Mesh Network
UDP	User Datagram Protocol
UMIC	Ultra high-speed Mobile Information and Communication
UMTS	Universal Mobile Telecommunication System
UV	Ultraviolet
VCS	Version Control System
VLAN	Virtual Local Area Network
VM	Virtual Machine
VMM	Virtual Machine Monitor
VoIP	Voice over IP
WAN	Wide Area Network
Wextool	Wireless Experimentation Tool
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network
WMN	Wireless Mesh Network
WSN	Wireless Sensor Network
XEP	XMPP Extension Protocol
XML	eXtensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
YAML	Yet Another Markup Language

Chapter 1

Introduction

The internet, and networks in general are in many ways part of everyday life. As their usage increases, their faults and limitations begin to increase, leading network researchers to evaluate and develop new solutions to overcome these limitations. New network technologies must emerge, to overcome these new problems and to avoid those in the future. These problems vary across a wide range of networking areas, from traffic optimization in telecom operators, scalability in a wide area network or broadband multimedia streaming. For researchers to create new network solutions, they must first test them accordingly and prove them consistently, before it can reach a larger community.

Simulations are an inexpensive and controllable method of evaluation, when making initial approximations to new technologies. However, simulations rely on simple network models and do not represent accurately the complexities of a faulty and dynamic environment. Furthermore, such scenarios are too flexible, and normally not taken in consideration by simulation based scientific studies. These problems have lead researchers to question the reliability of such results. Emulation is an alternative solution to simulations, more importantly in the implementation phase. It consists of the representation of a larger number of components, with fewer physical resources then those required. This solution fails also, when attempting the reproduction of more faulty environments, as their complexity increases. Consequently, experimentation is only trustworthy when tested under real-world conditions. Experimentation platforms must be developed in order to make this happen. The network community refers to these platforms as testbeds.

Testbeds are used in many network areas, and typically leads to more mature network solutions, resulting from accurate tests made by these facilities. Global initiatives have been created which have dedicated themselves to the improvement of testbeds, by studying scenarios and operating methods of driving testbed facilities. Their creation raises many development and maintainability problems. Usually a testbed is created within the scope of a project, and its construction involves high deployment and operation costs. Once the project is complete, the testbed falls into disuse, resulting in a waste of resources. Furthermore, testbed operation is very difficult, typically lacking instrumentation and maintainability capabilities. It is difficult to manage testbed resources and provide automated and consistent ways to create and reproduce scenarios. To assert existent methods of driving these testbeds is a key topic of this dissertation, so as the proposition of a solution to enable the reuse of these facilities, while proving itself as an important tool for network test.

1.1 Goals

Different testbeds imply different ways to manage resources, and usually different types of scenarios. Instrumentation between user and testbed resources are part of the state-of-the-art. In the early stages of testbed creation, operation could be reduced to a custom deployment of operating systems, and tests executed manually. Experimentation on testbeds lacks efficient and productive ways to create these tests, while the testbed owners lack in administrative maintainability of such facilities. The mechanisms necessary to use and operate in a testbed are the main subject of this thesis, the main topics of which are:

- To learn about the different network test approaches
- Gathering past and current testbed deployments
- To study the management tools of some of these testbeds
- Ways to improve general use and instrumentation for users and operators

From this study, it is absolutely necessary that we identify important requirements, to build a platform that will better adapt to users. The development of this platform will be supported by a currently deployed testbed at the Instituto de Telecomunicações.

1.2 Outline

Chapter 2 explains the three different approaches to network testing, explaining tools and general architecture; Simulation, Emulation and Testbeds are all presented. This section focuses on the overall testbeds, having a special focus on their management mechanisms, scenarios and related architectures. They have been separated into two distinct groups: those with a clear management infrastructure, and those without. Since there is a large number of testbeds, the managed ones have received greater focus. Of all the different testbeds and tools considered, generic tools are practically non-existent.

Chapter 3 presents a more in-depth explanation of the architecture, along with the current limitations of the only state-of-the-art tool related to testbeds instrumentation. Furthermore, we will also describe the testbed deployment which will support the work of this thesis, explaining its architecture and capabilities.

Chapter 4 explains the identified system requirements, and an extension is proposed to address some known limitations of testbed deployment while aiming to improve overall testbed infrastructure. Here an implementation will be explained, from its major features to developed modules.

Chapter 5 discusses an overview of this testbed usage, as well as the workflow and validation of the presented extension. We will describe different scenarios, from simple to more complex experiments. Performance tests are made to prove this solution's scalability from a web application point of view.

Chapter 2

State of the Art

Since the establishment of the Internet in society, new technologies have emerged for community usage. These technologies range in numerous areas: Quality of Service (QoS), Voice over IP (VoIP), File sharing, Internet Protocol Television (IPTV), etc. They experienced several iterations in order to support current usage, as well as the development of new solutions. Before being presented to a larger community, these solutions must be accurately tested and validated in order to assure a major adoption. This involves a process of creating the idea and testing it theoretically and mathematically, developing prototype implementations and stress tests under real world conditions before it can become available for production. The process is composed by several phases:

- An idea is created, and its algorithms and behaviors must be observed and correctly validated before implementation takes place. Simulation tools are used in this process.

- When this idea is validated, implementation of a functional prototype takes place. In this phase tests are made in a more isolated and realistic environment, for debugging purposes. Emulations tools are used here, and present higher accuracy compared with simulation software.

- When the implementation reaches a higher level of stability, it needs to be testbed under real world conditions to become a bulletproof solution. Testbeds are deployed for this purpose, and are needed to prove a solution reliable when deployed in production. Tests are done under an uncontrollable environment, where software, hardware and external constraints are present. In wireless networks this is more verifiable due to external interferences and medium issues that might arise and compromise solutions.

In this chapter it will be studied these network approaches, to acquire the background and knowledge of their limitations, before the study of testbed deployment and tools is made. It is important to have an understanding of how simulation and emulation works, in order to better approach testbeds manageability.

2.1 Simulation tools

Simulation software is well known, not only in computer science areas. This approach is widely used many scientific areas, from biology to aerospace industries. This kind of software provides inexpensive and agile ways to test an idea under very specific conditions.

In computer networks, simulators are important to make first approximations of the base idea. Users define their network, implement protocols and describe scenarios using a simulation framework. Most of the simulators maintains track of all events during the experiment. Simulation time is not linear, and moves forwards through the dispatch of events. Events mark a time instant and a change of the "system". These types of simulators are called discrete event simulators. Others direct their efforts towards scalability representing large networks. All share one characteristic: the network model. The model consists of a detailed description of the network: the physical layer, traffic flows, routing queue mechanisms, etc. This is necessary because simulations reside only on software, so a precise network model is directly related to the accuracy of results.

A survey to existent network simulators is made in this section. Besides open-source and free simulators, it is presented two simulators which are proprietary solutions. Only an overview is made on these last simulators, since their internal architectural cannot be accessed.

2.1.1 Network Simulator

Network Simulator (NS) is one of the most used simulators by the computer networks research community, being free software and licensed under GNU General Public License Version v2.0 (GPLv2). It consists on a discrete event simulator, whose time progresses with the release of events managed by a scheduler. NS is supported by a large community, and it has been extended with new applications and protocols. At the present time, there are two maintained versions of this simulator: Network Simulator 2 (NS-2)[1] representing the legacy version of the original project, already with many contributed extensions in a large variety of protocols; and NS-3[2] that aims to replace the past version with a refactored and much improved infrastructure.

NS-2

NS-2 architecture follows a modular approach to obtain both extensibility and scalability, and achieves this by dividing the data operations from the control operations. A simulation can be defined as an Object Tool Command Language (oTcl) script, containing configurations and topology information (among other information), which is passed to the engine, generating the results. Control operations are passed in this oTcl script and are handled by an embedded interpreter in the core engine. Since almost all interpreted languages have a bridge with C++, the core engine exposes some of its modules (e.g. traffic generators and receivers, routing protocols, error models to simulation link and node faults), to the oTcl engine, promoting a symbiosis of both languages, sharing objects between them. This approach simplifies all maintenance using a more high-level language instead of a more complex one, such as C++. Data operations are carried out by modules in C++ in the core engine, as they require a much faster response, and to avoid simulation errors. These operations are called per-packet actions, varying from traffic generation to trace output generation.

NS-3

In comparison with NS-2, NS-3 has had the entire infrastructure refactored, promoting modularity, loose coupling and an improved object model to create simulations. Future integration with testbed/emulation infrastructures is envisioned, to provide users with easy migration between network test environments, and to include wireless network models better suited to current needs, such as Worldwide Interoperability for Microwave Access (WiMAX), Code Division Multiple Access (CDMA) and other emerging standards. These features do not exist in NS-2 version. Additional efforts were made to implement network protocol stacks as realistically as possible. This is done through the Object Model, where the base functionality of the simulation is located. Here are implemented the primitives related to packet exchanging between nodes. As seen in Figure 2.1, a `Node` is described by `NetDevices` which communicate with other Nodes via Channels. Applications communicate using interfaces that are defined when implementing or using a network protocol (e.g. Transmission Control Protocol (TCP)/Internet Protocol (IP)). The API's for these interfaces are similar to real-world implementations, existent in operating systems (UNIX, Win32 native API's). This Object Model was made in such a way as to promote the reuse of existent components, with different implementations, without recompiling or modifying core libraries. In NS-2 this was a drawback, since classes could not be reused to provide additional functionality, requiring the developer to modify base classes and recompile the entire NS application. Simulations are defined in C++ with an optional Python interface. During simulation run time, tracing and statistical data generation are provided, which can later be displayed in *Wireshark*.

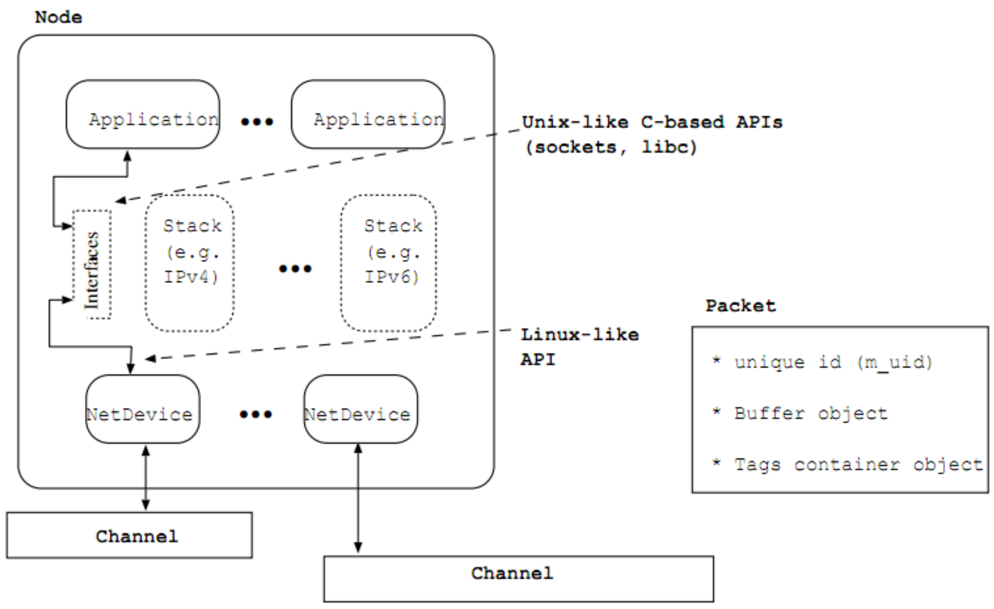


Figure 2.1: NS-3 Object model

2.1.2 OMNeT++

In comparison to NS-3, the Objective Modular Network Testbed in C++ (OMNeT++)[3] does not provide the whole network simulator package. Instead, a framework was developed, following the same discrete event simulator principles. OMNeT++ relies on the creation of simple modules to represent the behaviour of the model, which can be linked together as compound modules to define the network. It uses a specific language called Network Description Language (NED), for the interaction of the modules and network setup. This framework is extended with modules or NED extensions to support the creation of simulations. The modules will be compiled and executed along with the simulation kernel (the event scheduler), similar to what happens in NS-3 simulations. The simulation configuration is clearly divided from its modules, and since it promotes loose coupling, it can be easily embedded in other applications. OMNeT++ is a strong environment also with a strong Graphical User Interface (GUI) support, and an extensible architecture with a continuously growing community. Compared to NS-3 the main difficulty is the creation of simulations, since this is only a framework whereas components must be created and combined to form a simulation. Fortunately, there are OMNeT++ component packages[4], which facilitate simulation and components creation.

2.1.3 JiST

A different approach to network simulation was taken by Java in Simulation Time (JiST)[5], which in compliance with its name allows the implementation of network simulations in standard Java. Network Simulations are normal Java programs running in a custom Java Virtual Machine (JVM), in which the kernel responsible by the simulation resides. The user is given access to an Application Programming Interface (API) to develop their simulations. JiST will modify generated byte code to introduce time semantics in the simulation.

It takes advantage of JVM features, such as garbage collection, reflection, type-safety, and cross-platform, and allows users to reuse their Java network applications into the simulation. Furthermore, it provides simulations with an object model, which optimizes concurrent and blocking operations, providing high performance over other simulations approaches.

Although the documentation is somewhat out-dated, JiST is mostly used in conjunction with Scalable Wireless Ad hoc Network Simulator (SWANS), a simulator for mobile ad hoc networks built on top of this engine. SWANS is a package of components built on top of JiST infrastructure, to provide similar functionality to other network simulators such as NS/OMNeT++

2.1.4 GloMoSim

Global Mobile Information Systems Simulation Library (GloMoSim)[6] is a simulator based on the Parallel Simulation Environment for Complex Systems (PARSEC) [7]. It is able to create simulations that scale to thousands of nodes, with similar resource performance to the JiST. It consists of a parallel simulation environment using PARSEC as a middleware language to describe simulations based on their parallel execution model. These are discrete-Event Simulations modeled via a Message-based approach, where physical processes are modeled via Entities and communication between them is made via time stamped messages. The language is used as the interface with the scheduler. Simulations written in this language are translated to C by the PARSEC compiler, which is in charge of arrangements between entities and the simulation kernel, to create the simulation.

This environment can be used in general simulations, it is not restricted to network ones, providing a higher performance factor when compared with NS-2, able to reproduce much larger networks (e.g. 100 000+ nodes)[6]. GloMoSim consists of a library using PARSEC as a language to implement network protocols of the different layers similar to the Open Systems Interconnection model (OSI) model. API's were created to interact between all layers, ranging from radio, Media Access Control (MAC) layer, ad-hoc routing protocols or the TCP/IP stack.

2.1.5 SimPy

The Simulation in Python (SimPy)[8] is a discrete event simulator package entirely written in Python language. It is best known for being the only Python package in discrete-event simulators, as well as for its ease-of-use and extensibility. Since it is a general purpose simulator, it is used in many scientific areas, such as computer performance, epidemics, space surveillance, transit systems, etc. Similar to GloMoSim, it follows both a Parallel Execution model (Process-based), and Object-Oriented principles. `Processes` define the main active entities in the simulation, and `Resources` are shared among processes. Python `yield` calls are used for event triggering and process control of the simulation. Similar to NS-3, SimPy will trace all processed events to be displayed later. All this monitoring is optionally enabled through a `Monitor` entity, which collects tracing and statistics. Later on, this data can be displayed in real-time or at the end of the simulation, by using SimPy Plot or Real-time modules. To facilitate simulation control, it comes with a GUI framework, for users to easily change parameters, visualize results and control simulation execution.

SimPy introduces a software package enabling an easy development of discrete-event simulations, taking advantage of their simple API and Python fast learning curve. Being a general purpose simulation package, the main drawback of this approach is the non-existence of network models, to support network simulations.

2.1.6 GTNetS

Georgia Tech Network Simulator (GTNetS)[9][10] is another C++ software, similar to NS-3, known for its scalability when reproducing large scale networks (approximately 100000 nodes), with added support for distributed simulations. The entire simulation topology is distributed into smaller models that run on separate simulators. This has many advantages, being able to reproduce bigger scenarios, and reducing simulation complexity into smaller parts. Similar to NS-3, GTNetS focuses on delivering network protocols stacks as implemented in real-world. Protocols from different OSI layers are clearly separated; for instance, a packet must extend `L2Protocol`, `L3Protocol` to create compatible Link or Network Layer protocols, respectively; `Interface` defines a network card, similar `NetDevice` from NS-3. Simulations are C++ applications supported by a whole set of network protocols stack.

2.1.7 QualNet

Similar to GloMoSim, QualNet[11] is a distributed and parallel-based simulation, used for modeling large scale networks. The simulation engine leverages the parallelism achieved by GloMoSim to scale to multi-core processors and supercomputers, providing faster simulation times and even bigger network scenarios. Another strength of this simulator is being cross-platform across all mainstream operating systems. It provides a rich graphical users interface, where main components consists of a scenario designer to create the simulation, changing the location of the nodes, network nodes parameters, physical parameters, etc. Starting the simulation, the user is able to observe the generated metrics, packets structure, traffic flows and view dynamic graphs of these results, using PacketTracer and Analyzer components. Scenarios can be later animated using Visualizer and Animator components. Network Models can be either extended by the user, or take advantage of QualNet model libraries.

2.1.8 OPNET

OPNET[12][13] is another well-known commercial solution for network simulations. It is a discrete-event simulator, with highly detailed network models, and data analysis tools to analyze all the simulation data, with graphics supporting their visualization. OPNET is divided into a modeler, planner and a model library to create simulations. Network models have high level of detail, thus providing more reliable simulations. A rich GUI also supports simulation creation.

2.2 Emulation tools

Simulations usually lack integration to test implementations. As simulations began to evolve, their credibility also began to be questioned. A network model, in which simulators depend, lacks several details necessary to correctly reproduce scenarios. Emulation focuses on cost reduction and optimized ways to test and developed network related software. Emulation runs developed applications in real machines, in a smaller scenario compared with a real world use case. It is mainly described by simulating the links between the nodes, usually implying one machine to represent many nodes in a topology. Emulation is a middle ground between simulation and testbeds, while maintaining a flexible environment in terms of dimension and parameterization; it also provides the reliability of a testbed and reduced cost of deployment.

In the following sections, current options in network emulation are studied, from their architecture to their main features.

2.2.1 Netbed

Netbed, originally “Emulab”, is one of the largest experimentation facilities in network research that focuses on emulation capabilities, to bring “real-world” experimentation to users together with the simulation environment control and ease of use [14][15]. It consists of a space/time-shared platform providing experiment management, along with collaboration enforcement between users.

Different types of resources power Netbed’s facility: clusters sustaining dozens of pc’s to support its large-scale emulation (nearly 168 PC's in one cluster); distributed nodes outside the main facility, namely wide-area nodes; and simulated nodes employed by Network Simulator Emulation (NSE)[16] to enable scalability beyond the limits of Netbed's physical resources. Netbed acts almost like as an experimenter's virtual machine, providing services in terms of resource and link allocation, experiment management and software deployment. It almost emulates an operating system for network experimenters. Users access the Netbed platform, specifying an NS script, which will contain application configuration, execution control (e.g. traffic generation, software deployed), and topologies. Topologies contain nodes, links configuration needed to reproduce their experiment, where it will be "compiled" to the hardware facility. The nodes specified in the experiment, can be either local nodes, simulated or distributed ones, whereas links between them are mapped directly, or emulated, via Dummynet[17] which emulates the bandwidth and packet loss. Experiments behave almost as processes within an operating system. The experiment script serves as the program, whereas the overlay network is the resource used. The "OS" will then preempt experiments, "swapping" its execution as a process scheduler does. Netbed is best viewed as an overlay network, allowing users to transparently allocate resources, much like a virtual machine abstracts operating systems of their target architecture instruction set.

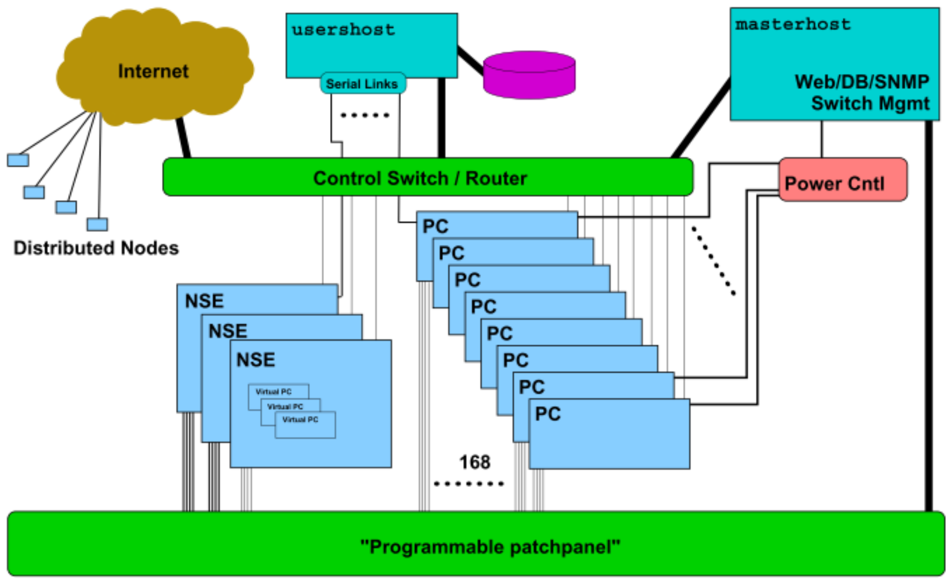
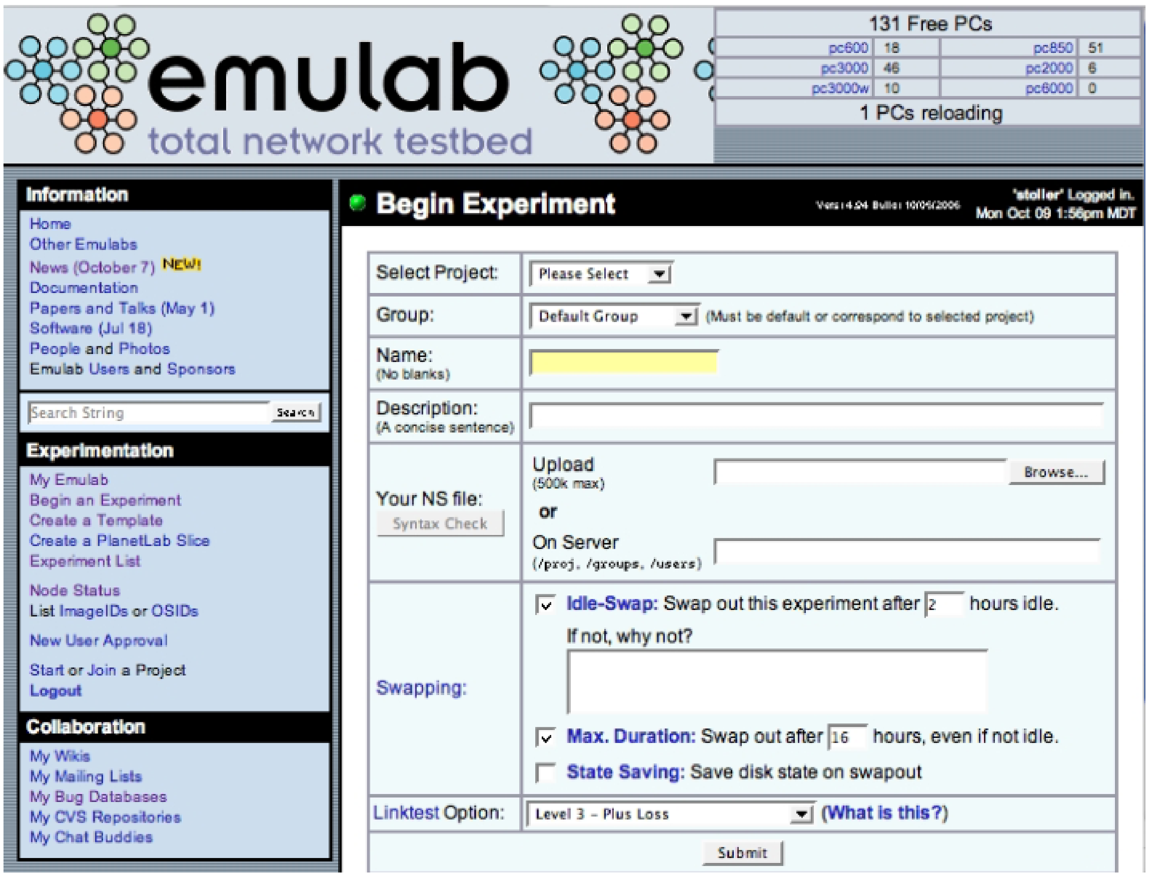


Figure 2.2: Netbed Architecture

Clusters supplying the local nodes are interconnected with high-end switches. Each local node has five interfaces: one for the control network and the other ones for experimentation. The "programmable patch panel" illustrated in Figure 2.2 refers to the network resources programmability, whereas Virtual Local Area Networks (VLANs) are established in order to create well-isolated topologies and Dummynet is deployed in local nodes to emulate bandwidths and loss. `usershost` supply users with home directories to store operating system images and scripts. This module is also the synchronization point between local and distributed nodes. `masterhost` is the authorization and management entity maintaining information about all nodes, monitoring data, web server for the portal, etc. All resources can be accessed with root privileges by its users. Access control by users in distributed nodes is made through FreeBSD Jail mechanism, which is a primitive form of a virtual machine. If users corrupt resources, Netbed's infrastructure is able to restore disk images quickly, since all information on experiments is maintained by the `masterhost`, enabling restoration of an experiment's state. A tool called Frisbee[18] has been created to enable these disk restore mechanisms.



131 Free PCs			
pc600	18	pc850	51
pc3000	46	pc2000	6
pc3000w	10	pc6000	0
1 PCs reloading			

Information
Home
Other Emulabs
News (October 7) NEW!
Documentation
Papers and Talks (May 1)
Software (Jul 18)
People and Photos
Emulab Users and Sponsors

Search String [Search]

Experimentation
My Emulab
Begin an Experiment
Create a Template
Create a PlanetLab Slice
Experiment List
Node Status
List ImageIDs or OSIDs
New User Approval
Start or Join a Project
Logout

Collaboration
My Wikis
My Mailing Lists
My Bug Databases
My CVS Repositories
My Chat Buddies

Begin Experiment Vers: 4.24 Build: 10/05/2006 'stoller' Logged In. Mon Oct 09 1:56pm MDT

Select Project:	Please Select
Group:	Default Group (Must be default or correspond to selected project)
Name: (No blanks)	
Description: (A concise sentence)	
Your NS file: [Syntax Check]	Upload (500k max) [Browse...] or On Server (/proj, /groups, /users)
Swapping:	✓ Idle-Swap: Swap out this experiment after 2 hours idle. If not, why not? ✓ Max. Duration: Swap out after 16 hours, even if not idle. ☐ State Saving: Save disk state on swapout
Linktest Option:	Level 3 - Plus Loss (What is this?)
	[Submit]

Figure 2.3: Netbed Portal Screenshot

Portal On top of this emulation infrastructure, Netbed provides ease of use to users and organization through their portal, in which all the experiment configuration takes place. They refer to this environment as an experimental workbench[19], where users have an organized manner to create and store their experiments, and taking advantage of Netbed's experimentation isolation and repetition, to support scientific studies. A user is able to access Netbed by requesting a project through a web form. Once the project is approved, access to the Netbed is granted and users are able to assign other members. They create experiments through the portal, and observe related virtual topologies and node link properties. Once an experiment is created, access is granted to the allocated nodes. An experiment is defined by a template which is the main data repository, containing the NS script, binaries of the system under test, files, or any additional data which describes the experiment. This template is instantiated with resources and parameters assigned. A run is the execution of that instance during a specific amount of time. Output files are collected during run and stored for further analysis. Records are data repositories related to an experiment run. Filesystem monitors and packet capturers, support the experiment with relevant data to be further analyzed. Users can report bugs on their experiments and collaborate with other project members to compare results and fix experiments scenarios in the course of a study.

Netbed is an example of an experimentation platform, featuring an organized way for network researchers to conduct their studies obtaining relevant results. It is one of the most known platforms in experimentation, covering simulation, emulation and real-world approaches. The only drawback encountered was the lack of mechanisms to get results data or obtain custom metrics for an application of a system under test. Furthermore, it relies on expensive infrastructure and dependence on the `masterhost`.

2.2.2 StarBED and SpringOS

Similar to Emulab, StarBED[20] presents a large-scale network testbed consisting of 512 PC's interconnected by intelligent switches. Each PC's runs ten virtual machines which allow an experimenter to create topologies using up to 5120 nodes. It has been used in a high variety of experiments: emulation of wireless networks and cellular networks; TCP behaviour; multicast traffic benchmarks; multiplayer online games evaluation.

StarBED's structure is similar to Netbed, differing in scheduling and reservation mechanisms. In Netbed, nodes are chosen varying from their availability, and experiments swapped if inactive. Using StarBED, nodes are immutable and thus not reutilized by another experiment, until its reservation time slot expires. Like Netbed, the key of this testbed is to provide users with large topology representations and the means to support experiments. SpringOS is the software package which powers the testbed with experiment management procedures. It consists of a collection of modules that are responsible by the nodes software and topologies management. The user issues a configuration file that is loaded by the *kuroyuri master*, which is the core function driving experiments on the testbed. This file contains network topology and information related to the desired scenario to be reproduced. The *kuroyuri master* will ask for the resource manager to assign nodes to an experiment, if available. Node power management is made through a Wake-on-Lan agent, located at the Switches, which instruct nodes to power up. Then *kuroyuri master* will tell a Node Initiator located each node, to download the image to be booted afterwards using Pre-eXecution Environment (PXE). Topology is created using VLANs, according to the information on the configuration file. This file also contains semantics for users to specify actions during the experiment execution. *kuroyuri master* will handle these execution semantics.

2.2.3 ModelNet

Modelnet[21] is a large scale emulator, focusing on the scalability and realism in the reproduction of distributed network environments. It allows applications to run on unmodified operating systems across a variety of network scenarios, such as Peer to Peer systems, content distribution networks, distributed file systems, transport-layer protocols, etc. The principle of operation is the transparent use of this emulator to allow unmodified systems/applications to be used in large emulated networks. Testers deploy their applications on edge nodes (as referred by them) through core nodes running the target network. ModelNet captures all packets in the network to a queue system, where they are removed according to an input bandwidth, latency or loss rate. The great feature about ModelNet is their ability to create large-scale networks using this method.

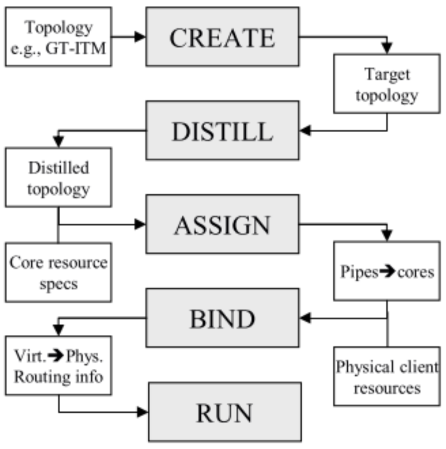


Figure 2.4: ModelNet Architecture

Users create topologies, by issuing a Graph Modelling Language (GML) file, using specific applications (e.g. Georgia Tech Internet Topology Model (GT-ITM)). These topologies are analysed for further simplification (if possible), to reduce emulation costs upon reproduction. This is referred as the "Distil" phase. Next, the "Assign" of the distilled topology takes place: varying from the number of core nodes it creates pipes with specific link properties to emulate the network, as illustrated in Figure 2.5. To create these pipes, ModelNet uses an extended version of Dummynet[17] which emulates this queue behaviour with bandwidth and loss regulation. The final phases, namely "Bind" and "Run", refers to the binding of the virtual network (previously mentioned pipes), to the physical network, by creating scripts to automatically configure the edge and core nodes. The edge nodes running targeted applications must point to the corrected emulated nodes, instead of their default addresses.

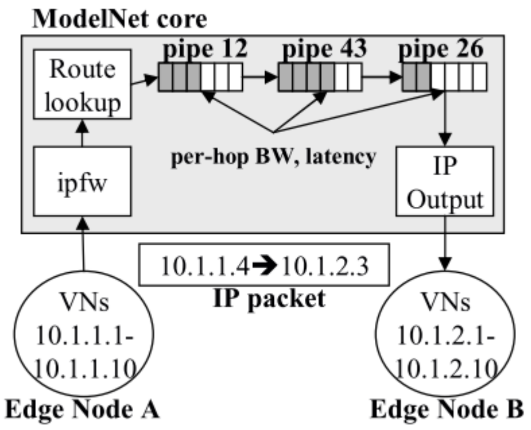


Figure 2.5: Pipes between edge nodes, creating the ModelNet emulated network

2.2.4 Netem

Netem[22] introduces a packet filtering mechanism through a queue behaviour, similar to Dummynet. Two interfaces manage all the process: while one queues time stamped packets to be sent, another releases them to the network device. The user has a command-line tool called `tc`, where he specifies the network interface to be emulated and the parameters such as delay, packet loss or traffic control into separate queues. Netem has many different options to control traffic flow, although their limitations rise when trying to emulate more complex systems. Netem adapts better to single flows, due to the lack of parameters to represent complex networks, as stated by the authors. One of the advantages of this emulator is its ease of use.

2.3 Live Experimentation

Some of the most well known simulators in the network community were studied. Most of them are in active development, and already enjoy a large community. Simulations tradeoff computational power for simulation time, which depends on a representation of the network, here referred to as network models. These models must match a precise representation of the network, normally not taken into consideration when carrying out simulation tests. Furthermore, a simulation withholds a high number of configuration parameters that are usually not taken into consideration. Scenarios are too flexible and simulations cannot predict or recreate all these scenarios[23]. This leads to questioning of the simulations reliability, and suggests that new alternatives should be adopted to support their studies. Nevertheless, simulations are an important first approximation, providing the most controllable environment, and an inexpensive manner to reproduce network tests. As solutions begin to enter prototype stages, exhaustive tests need to be made, under real-world conditions. Testbeds are deployed to support this kind of environment, consisting of complete hardware infrastructures. Optionally, tools can be installed helping users to make better use of a testbed. In the next sections, it is presented different types of testbeds, dividing them in two groups: unmanaged and managed. In the scope of this survey, sensor testbeds and federation of testbeds are not included.

2.3.1 Unmanaged Testbeds

Most early deployment of these testbeds did not present management tools, having its users to manually conduct their tests. In this section such examples are shown, encompassing their network area and objectives. These deployments were catalyst to more developed testbeds that came after.

RoofNet

Massachusetts Institute of Technology (MIT) RootNet[24] consists of a 37-node mesh network, spanned in a four square kilometer area, focusing on the study of community networks in an urban area. These community networks are created in two ways: to build a multi-hop network with high-quality links in chosen locations; or “hotspots” distributed by a set of individuals. They claim that unifying both approaches would minimize maintenance costs while providing a good wide-area coverage. Testbed nodes are located on the rooftop of the buildings being hosted by volunteer users. Each node runs a OpenWRT software distribution, deployed with a web server for real-time status. The primary feature of this testbed is that it requires no additional configuration or planning to deploy a node.

Hyacinth

Hyacinth consists of a 9-node mesh network, composed of conventional PC's with two 802.11a network interface cards. Their research focuses on studying multi-channel approaches to improve overall network throughput. Distributed algorithms were developed to dynamically assign channels to route packets, according to the traffic load. They believe that by dividing traffic load between wireless channels, they can improve network traffic by an order of 6 to 7[25]. Their network architecture is divided into a mesh network core connected through wired connectivity gateways. Each mesh-router is equipped with a traffic aggregation device that is responsible for gathering traffic between different channels and sending them to the clients. Each mesh router connects to another using a different channel. They made their study first using simulations and later on this testbed, showing significant bandwidth improvements.

ScaleMesh

ScaleMesh[26] focuses on reproducing large networks emulated in a much smaller space, namely a miniaturized testbed. Through signal attenuators they decrease signal power to limit the transmission range of each node. This kind of approach enables a more efficient evaluation of network protocols in networks with several hops. The testbed deployment consists of 20 nodes, with a 802.11b/g PCI card, built in a 10m x 6m area. This testbed focuses on the study of multi-hop routing protocols, such as the Optimized Link State Routing Protocol (OLSR). A successful example of this kind of testbed is explained in the next section.

2.3.2 Managed Testbeds

In this section managed testbeds are described, explaining the approaches in the different deployments, with a special focus on their management tools. This approach is important to get aware of different approaches in how to manage and use a testbed.

WiSEMesh

Similar to the RoofNet testbed, Wisemesh[27] is an unplanned and auto-configurable Wireless Sensor Network (WSN), consisting of 56 nodes distributed over the Korea Advanced Institute of Science and Technology (KAIST) campus area. It was deployed to provide a platform to support network tests while serving as a production network for students. WiSEMesh aims to build a network management system, to support the whole platform with easy extensibility. It supports different types of radio interfaces, and support for sensor networks. The whole infrastructure was built on top of open-source software/drivers, and holds no specific hardware requirements.

The Wivi software architecture consists of a set of modules divided in two entities: the management server and the WSN node. A Node Agent is deployed to enforce the automation process of software updates and configurability. The ones in charge for the nodes are the management server, which periodically gathers information about nodes, and a centralized database in which such information is stored and that will maintain all network information. Furthermore the server provides a web user interfaces for: network status visualization; software updates management and nodes configurations.

UMIC-Mesh

UMIC-Mesh[28] is a testbed with 52 identical mesh routers distributed across two buildings of the department of science of the Rheinisch-Westfaelische Technische Hochschule Aachen University (RWTH) Aachen University, from research work done by the Ultra high-speed Mobile Information and Communication (UMIC) Group. Following an hybrid approach, it combines the best of two worlds: having real hardware which implies having realistic results compared to those obtained from simulations and the easy deployment and debugging achieved with the virtualized environment supporting the mesh routers. Most of their research work was created when studying the issues related to the TCP, focusing on network protocol evaluation and tools. They developed a tool similar to *iperf*, called *flowgrind* with the same purpose but with an emphasis on WSNs. This tool is able to test against multiple servers, feature that currently the previous mentioned tools are found lacking.

The 51 nodes are divided in three layers: three gateways with wired connection to Internet, supported with a set of routers that provide Internet to the next layer, which are representing the "clients". The clients are divided in two groups: routing mesh clients being the ones communicating in a multi-hop fashion and the non-routing mesh clients, in the conventional way. The virtualized environment is located between the Internet Backbone and clients for the tasks mentioned previously. For deployment, a system image is provided to each node with Network File System (NFS), and booted via network using PXE with EtherBoot.

The node configuration is too decentralized and the testers are required to have knowledge of the deployed tools, resulting in a bottleneck to the management of the testbed. Network monitoring is achieved using Network Simple Network Management Protocol via Web (Net-SNMP), providing some control on the network infrastructure and some capacity to change parameters such as transmitting power or wireless channel. No additional tools are provided to enhance the repeatability, reproducibility or evaluation of the test scenarios. For now, their research work focuses on TCP and since *flowgrind*[29] already collects most of the metrics; perhaps no additional effort need be applied to enhance experimentation on the testbed.

Ad-hoc Protocol Evaluation

The Ad-hoc Protocol Evaluation (APE)[30] project from the Uppsala University was a testbed whose initial purpose was to study ad hoc routing protocols on real world environments. Since it focuses on mobility, the testbed has no persistent structure. Furthermore, the project focuses on easy deployment on the nodes, repeatability and easy creation of the experiments. Being easy to deploy, the participants of the experiment only need to install an APE distribution on their computers. This software package was created assuming that the participant would not know the tools and scripts provided (e.g. students), ensuring easy installation for anyone.

The APE distribution is an execution environment consisting of a Linux distribution based

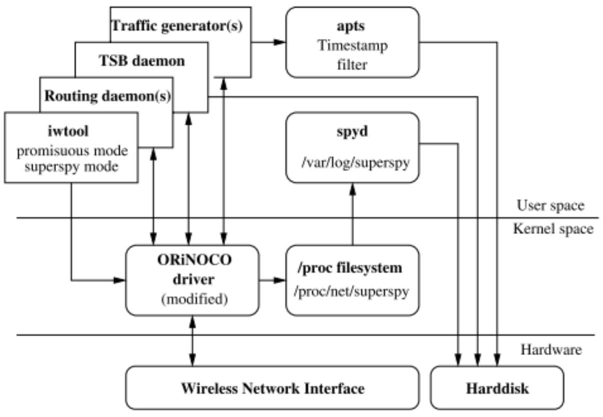


Figure 2.6: APE software architecture

on Red Hat Linux with a set of scripts that enable test runs, data gathering and post-analysis tools (Figure 2.6). The software package can be customized and compiled from the source. Customization includes the selection of a set of routing protocols implementations such as the well-known Ad hoc On-Demand Distance Vector Routing (AODV) and OLSR. One of the requirements is the use of ORINOCO 802.11 WaveLAN cards, due to some modifications made to the driver. These modifications include the logging on signal level and signal noise for all packets captured by a wireless interface, which is later processed to compute other metrics, such as the "virtual mobility" introduced by this project. Virtual Mobility usefulness is to determine the similarity of the test runs, and to serve as support for other metrics evaluation, such as packet loss.

APE also introduces the creation of scenario files to facilitate experimentation. These scenario files contain the movement and traffic generation patterns of each node, aas well as a sequence of actions (e.g. command execution). To enable experiments without any movement, APE distribution comes with MAC Filtering provided by the mackill tool, to emulate different topologies between nodes. MAC Filtering is also integrated within scenario files. When running an experiment, a "collect node" must be configured in order to synchronize and maintain all the log files, for post-analysis. Enabling the manipulation of the gathered data, a set of analysis scripts comes with the software package, providing access to: metrics such as the average number of neighbors per node or average Signal to Noise Ratio (SNR) per interval; graph generation of the results and also a GUI for log analysis.

Meshnet

University of California Santa Barbara Mesh Network (UCSB Meshnet)[31] is deployed in the campus with 25 mesh routers nodes, distributed over the five floors of a building. It is used in the research of wireless testbeds management over large physical areas. To achieve this they study the drawbacks of what they call "in-band" management and "out-of-band" management, and thus introducing a new concept. In-band management implies that management traffic is made part of overall testbed traffic. When configuring the network, issuing commands must be done in a precise and coordinated manner, so in addition with all the traffic, this might not happen, as it should. The traffic overhead introduced in the testbed and the management traffic is one of the drawbacks of having in-band management, introducing inaccurate results when doing experiments on the testbed. “Out-of-band” is achieved by creating a control network, only accessible on a wired connection, removing the management traffic out of the wireless network, and thus placing on a separate network. Although “out-of-band” management is hardly applied because the nodes aren’t always reachable by a wired connection (e.g. forests, or swamps). They took a different approach, implementing the ATMA framework, deploying along with the testbed node, a mesh network dedicated only to the management of the node. Their approach is showed using conventional routers, showing that is a cost-reliable solution.

Each testbed node is represented by two access points. One Access Point (AP) as a mesh node, and the other as a management node, supported with a mesh network, the nodes of which are called agents. Commands are issued through these agents, and carried over in a multi-hop fashion until arriving at the mesh node. Easy deployment is also one of the objectives of this research work when deploying the supporting WSN. The agents composing the support mesh network are self-configured: the managers announces themselves, and the agents controlling these mesh nodes, register themselves with the managers, to gain their control. The manager uses a beaconing system to make announcements in the network and synchronize configuration of agent network parameters. A set of tools is also bundled which helps to evaluate the interference in the wireless medium, and additional tools manage/monitor the network. The software stack was deployed and tested on conventional routers using an OpenWRT distribution, with a modified version of the AODV routing protocol, to help find the most reliable path to take in the network.

MiNT

The Miniaturized Network Testbed (MiNT)[32] from Stony Brook University is a wireless network testbed built in a small area, focused mainly on achieving mobile experimentation, hybrid simulation, continuous and autonomous operations. Here it will be described the initial project (MiNT) followed by its second iteration with mobile improvements (Miniaturized Network Testbed Mobile (MiNT-m)) and will finish with the third iteration, which tackles on infra-struture improvements (Miniaturized Network Testbed Mobile 2 (MiNT 2)). This testbed consists of a set of wireless nodes which communicate over multiple hops through their wireless interfaces. The key feature in this testbed is that it can be deployed on a small physical area (e.g. 4x2 meters). Through the attenuation of radio signals on the respective transmitters and receivers, space can be reduced and thus bringing a testbed in a much smaller space with easier setup and management. This approach reduces the interference with other wireless nodes placed in the same building.

The network () comprises a central controller node having multiple core nodes with mobile robots attached, representing the peers. The central node communicates with the core nodes within a dedicated wired network. The core nodes communicate with each other using wireless PCI cards with radio signal attenuators attached to an antenna, to reduce their radio signal as explained previously.

With MiNT the authors claim that using different attenuation levels does not affect the general network performance (throughput, signal quality variation, route discovery), and is able to provide the same reliable results as those with no attenuation. Since core nodes are

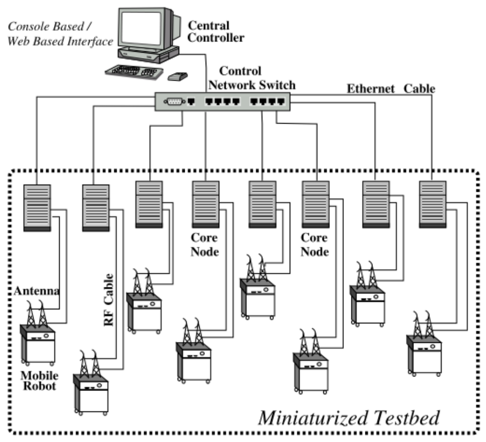


Figure 2.7: MiNT Network architecture

not mobile, the external antennas were connected to the robots, introducing mobility to the core nodes. The mobile nodes were created with low price, ease of assembly and remote controllability as key criteria. For this reason they used LEGO MindStorms[33] in the first version(Figure 2.3.2).

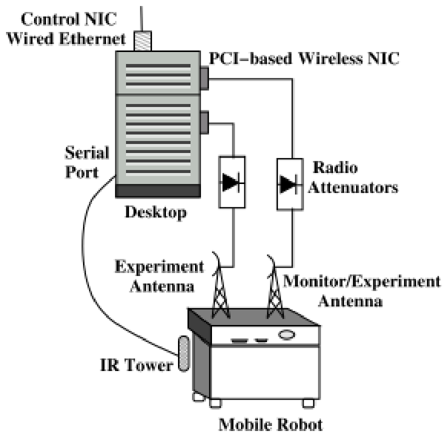


Figure 2.8: MiNT mobile robot

The controller is responsible for the statistics collection on the testbed and remote management of the nodes. It uses Simple Network Management Protocol (SNMP) for remote network management. Some SNMP components have also been implemented, to provide the ability to update the Management Information Base (MIB), with data such as transmit power and packet count. To avoid overhead in this experiment traffic, a wired control network was deployed dedicated to the management traffic.

To enable experimentation in MiNT, they developed a set of tools and a hybrid simulation. Among their tools, is provided a GUI to the experimenter. This application enables network topology configuration, by placing the nodes in ways satisfying the link properties for the experiment. Furthermore, this process can be automatic; topology constraints can be selected and the application will later calculate node positions and place them correctly. Furthermore, configuration scripts can be provided to enable mobility configuration such as node trajectories or even mobility patterns. Network Parameters and node remote access are also included in the GUI. Experiment Reproducibility is enhanced in the later versions, with features such as the modification of experiment parameters on-the-fly, pause experiments, simultaneous start/stop on all nodes. All traces can be collected during experiment, as well as an offline visualization of the packets transitions. Fault injection tools are provided for debugging, where the experimenter can create network faults such as packet drop, delay or corruption.

Although all these tools are useful, their main focus is the hybrid simulation. A simulator provides us with the complete control of some imposed environment, being able to change much of the network variables. However, the results obtained from these simulations differ greatly when using a testbed, due to the lack of detail in the simulation models. A hybrid simulation is a solution they have introduced by combining the ease of use and controllability of simulators with the reliability of the results when using a testbed.

Basically they achieve this by embedding the NS-2 engine within each node. An experimenter could pick up one of their scripts used in simulation and easily run it on MiNT platforms. To achieve hybrid simulations with NS-2, it is necessary to implement the Physical, MAC and Link layers, replacing them for the wireless card, firmware and real wireless channel. One of the few limitations is that one virtual node is associated with one physical node, consequently limiting the size of the network used in an experiment. They had to modify the NS-2 event scheduler in order to synchronize the simulator's virtual clock with the real clock, since all events were to happen in real time. They also had to reduce the number of events processed by the NS-2 scheduler in order to solve the problem of scheduling past events, and embed the virtual packets within User Datagram Protocol (UDP) packets, since they do not have the correct protocol headers for wireless transmission.

MiNT-m

With the second iteration of the MiNT platform[34], namely MiNT-m, efforts were mostly towards improvements of the mobile robot with also some improvements in the overall experimentation infrastructure. One of the problems encountered in the previous version was the mobile robot movement. Since there was a cable linking from the core node to the mobile robot, the mobility of the node was limited and cables between other robots could become entangled. As main requirements, they focused on were continuous operation, low cost, node mobility, node tracking and node control. They completely refactored the entire mobile robot, from its hardware to some necessary changes to the MiNT platform. First they removed the desktop nodes and created unbound nodes, enabling them to move freely without all the previous restrictions. Mobile nodes are now powered through Roomba Cleaners (see Figure 2.9)[35], along with the previously equipped attenuators, antennas and also with a new wireless powered board, called RouterBOARD[36]. This assembly powers a low-cost autonomous mobile node, although some other modifications were needed to enable automatic recharging and movement control of the node. Other software components were added to enable the following features:

- a tracking server that periodically captures images of the nodes in order to detect their positions;

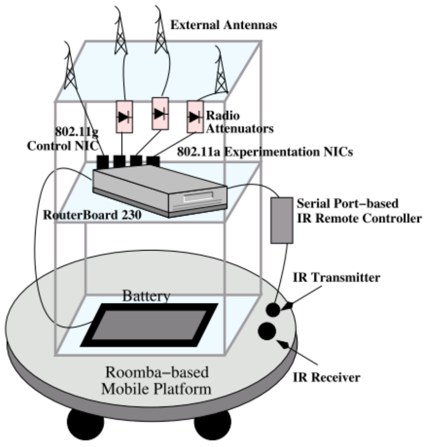


Figure 2.9: MiNT-m mobile robot

- a control daemon to register the positions updates from the tracking server and monitor the experiment;
- a daemon running on each node which communicates with the controller node to send movement commands, simulation events from other testbed nodes, radio-frequency monitoring and simulation control.

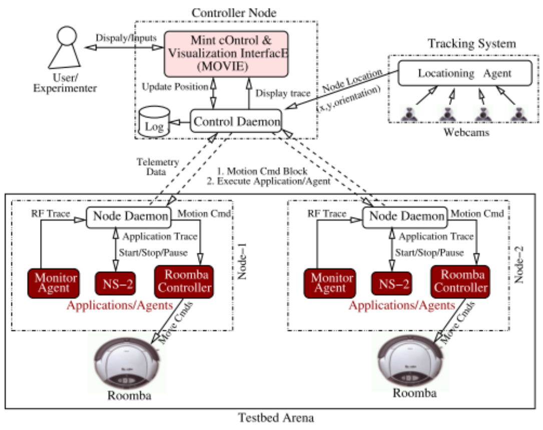


Figure 2.10: MiNT-m software architeture

The tracking server detects its nodes direction, orientation and location through a color patch included in all nodes. Trajectory determination is also carried out to avoid collisions. For continuous operation, the Roomba Docking station periodically sends Infrared (IR) Beacons, and each node checks its power, returning home to recharge if necessary. A set of spare mobiles nodes are provided in case some nodes are recharging.

The frontend application for experiments executions, now called Mint cOntrol and Visualization InterfacE (MOVIE), was also improved. This application was derived from the well-known Network Animator (NAM)[37], adding some key features: experiment pause/breakpoints, snapshotting/rollback and real-time status. To implement these features NS-2 scheduler modifications were need to be made. Some issues related to the real-time status were detected. According to the enabled tracing, this could affect the overall throughput, due to heavy traffic overhead.

MiNT-2

In the third iteration of this platform, major improvements related to node location and hardware improvements were added [38]. A need of a tracking system emerged due to some visual identification inaccuracies that may occur when lighting varied or cameras were moved. Sometimes their logical positions and the real positions in the tracking system were descalibrated, so they had to synchronize the robots periodically. In MiNT 2 a Radio-Frequency IDentification (RFID)-based tracking system was implemented, by spatially distributing tags across the testbed, to form a map. If one of the nodes crosses one of those tags, it would take that position. This approach eliminated the need to maintain a tracking server. Since RFID is an inexpensive technology, overall infrastructure costs were reduced.

DES-Testbed

The Distributed Embedded Systems (DES) testbed[39] from the University of Berlin, is one of the few featured DES TestBed Management System (DES-TBMS). The research focus in DES-Testbed is the comparison of simulations with real world experiments. It is one of the few testbed management systems, providing the necessary tools for users to manage their experiments.

The testbed comprises two different types of networks: DES-Mesh, which hosts 95 mesh routers equipped with 802.11 a/b/g adapters, called the DES-Nodes, with plans to extend up to 120 nodes; and DES-WSN with the same number of sensor nodes, establishing a sensor network at the same physical space. Both of the networks operate in parallel. The testbed server (DES-Portal) acts as a controller node, which is responsible for the management of the DES-Nodes and database support used by other components. Mobile nodes are any Wireless Local Area Network (WLAN) capable device, as well as smartphones.

The DES-TBMS software is divided into five components and four databases with the relationships, illustrated in Figure 2.11. Experiments are written using their domain specific language called DES Experiment Language (DES-Cript), DES Experiment module (DES-Exp) is the component where it maintains all the experiment descriptions and is responsible to schedule, manage and collect all data from experiments. The testbed can be monitored by DES Monitoring module (DES-Mon) during experiments, which will hold monitoring data needed by DES Visualization module (DES-Vis) to visualize the network state. Using the results from experiment and monitoring data, DES Evaluation module (DES-Eval) will evaluate all the data in order to compute Performance Metrics.

DES-Cript is a domain specific language based on eXtensible Markup Language (XML), to define and describe experiments to be executed on the testbed. With DES-Cript they aim to simplify experiment execution, therefore improving repeatability, enabling experiment reuse and rescheduling, along with the possibility of comparing multiple test runs of the same experiment. Figure 2.12 represents the experiment workflow in DES-Testbed, in which

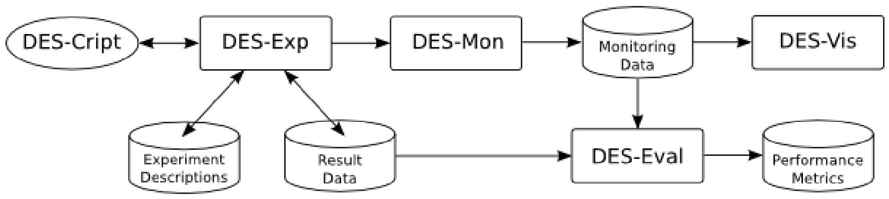


Figure 2.11: DES-Testbed Software Architecture

some steps are contained in the DES-Cript files: its configuration, execution and evaluation. These files contain general information, such as the number of iterations to make, description, author, start time. Groups can be created, which are sets of nodes, and commands issued to them. Preparation of the experiment such as the configuration of network parameters is issued in the init section of the file. There is a region of the file designated for execution, in which commands to groups can be scheduled. Evaluation scripts can be associated with actions, to process command output.

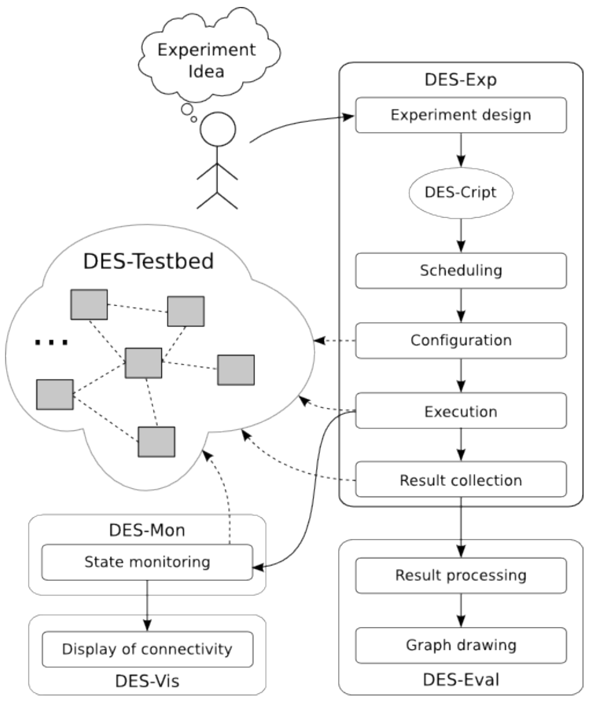


Figure 2.12: Experiment workflow

DES-Exp is the experiment manager, which creates, schedule and executes experiments. The start time of the experiments in designated by the user, and queued for execution by the manager. The user is emailed when results are available. When the experiment finish, output files are gathered and sent to DES-Eval. This module exists in two applications: one as the Web Interface, and the other serving as the engine which holds the previously described tasks.

DES-Mon is the testbed monitor, where node states are queried periodically using SNMP. The periodicity of the monitoring is specified in the experiment description file. The gathered data of each query includes all routing tables of nodes along with the state of the wireless interfaces. The routing information is then used by DES-Vis to display the connectivity between nodes. All changes in the connectivity can be animated through a timeline visualizing all networks changes in the entire experiment run.

DES-Eval automates the task of gathering all the log files to extract the important information for post-analysis. For this purpose, a set of evaluation scripts (or customized scripts), are provided to format the results. Based on the data, errors, averages, variance are also provided as statistical data for the desired performance metrics, included in the evaluation scripts. A set of these scripts is bundled with the module, enabling the usage of tools such as ping, ttcp and iperf.

ORBIT

Open Access Research Testbed for Next-Generation Wireless Networks (ORBIT) testbed [40] is one of the largest wireless testbeds, comprising 400 nodes, organized in a 20x20 grid. The framework, now known as the cOntrol and Management Framework (OMF), was developed to manage this testbed, outlining most of the current concepts of OMF. Their objective was to provide users, with an experimentation platform, with the necessary tools and services to easily develop their experiments. Along with the management framework, testbed usage control is made by a reservation service accessed from their portal where users reserve nodes for a certain time, to execute experiments on the testbed.

OMF is currently the state-of-the-art tool in terms of testbeds management and control. This framework was initially created by the ORBIT testbed, and later turned into a generic tool. This framework provides simpler means for users to create their experiments, while the testbed owners are provided with ways to manage testbed resources, ranging from power control capabilities, to complete node disk restore. This framework provides instrumentation capabilities for testbeds, distinguishing from other approaches described here. It is used by a large number of testbed deployments around the world[41]; they are refered as OMF-enabled testbeds. A more in-depth study of this platform is made in section 3.1.

NitLab

Network Implementation Testbed Laboratory (NitLab) is a wireless OMF-enabled testbed consisting of 35 nodes spanned in a six-floor building, with a focus on the study of wireless schemes and their performance. OMF enables the management of testbed resources, although not the most efficient use of them. To address this problem, they extend OMF in a way that experiments are scheduled according to their resources requirements. As described by the authors, the scheduling process is enabled through spectrum slicing[42][43]. In the NitLab reservation process, they specify the nodes, channels to use, and the time allocated to the reservation. Once this is complete, the testbed is configured with the booking information, and further access to nodes is provided to the user. The testbed maintains record of this data in order to avoid channel overlap, to allow concurrent experiments and a more efficient utilization of the testbed.

AMazING

AMazING is another deployment of an OMF-enabled testbed consisting of 24 nodes located in the rooftop of Instituto de Telecomunicações - Aveiro (IT). The testbed objectives are to support mobile experimentation, while providing a low cost general purpose testbed to create experiments. OMF support its infrastructure to provide the instrumentation, with some extensions and also some contributed patches. In the near future, this testbed aims to integrate NS-3 with its management infrastructure, inviting simulation adopters to take advantage of a testbed capability to provide trustworthy results. Their hardware infrastructure includes storage, and servers processing power to analyze results and support experiments. The extensions made to this platform were: custom power management through an ethernet interface; measurement infrastructure improvements, bringing a large set of metrics by using Internet Protocol Flow Information Export (IPFIX). Since this testbed supports the work of this dissertation, the AMazING deployment is thoroughly explained in section 3.2.

EXTREME

EXperimental Testbed for Research Enabling Mobility Enhancements (EXTREME) testbed is a multi-user experimentation platform to conduct wireless experiments[44]. This testbed consists of a generic platform sharing some principles of OMF and Netbed. They aim to reduce experiment creation time, for users to tackle more on their results and scenarios to reproduce.

The system architecture is inspired in Emulab and also provides similar functionality to XML used in ORBIT. A cluster of machines with two network interfaces (control and data), are interconnected using high-end switches/routers, providing access to Internet, and to the institution production network. All the nodes are controlled by a central server holding PXE and Frisbee configuration to allow custom operating systems to be deployed on nodes. They use a high-level language based on XML to describe the experiment. Users describe a physical topology and software to be used, which is later mapped to the nodes. Measurements are provided by means of a framework to help users to easily define the metrics to be captured. Underneath this framework, passive monitoring nodes interact with experiments resources to gather (only) network metrics, such as jitter, packet loss, throughput, etc. Each of these passive nodes will monitor a different experiment. Metrics are then sent to the central server, which will provide graphical results based on the retrieved information, packet traces and statistical computation of results. One drawback is the EXTREME testbed is the lack of application integration to generate custom metrics. Only network metrics are gathered, varying from what their platform supports.

The EXTREME testbed owners alleged the low scalability of OMF reproducing other types of wireless technologies. In this testbed they have also included support for General packet radio service (GPRS) and Universal Mobile Telecommunication System (UMTS)/High-Speed Downlink Packet Access (HSDPA) wireless networks. Their contribution to the network community is in the emulation and reproduction of a wide range of wireless technologies[45].

PlanetLab

PlanetLab[46] is a testbed with nearly 1,000 machines geographically distributed across the globe. These machines are all deployed with the same software package, namely MyPLC. This software package provides node control mechanisms, and a set of management tools which monitors node health, system activity auditing and also access/authorization control. It was designed to meet the need of researchers to understand and test new services and applications in a global scale. It was envisioned to serve the research community as a global overlay network[47], supporting long running services that might target a client community, through successive and seamless migrations.

The architecture follows five important principles:

- Distributed Virtualization: PlanetLab environment supports both short-term experiments and long-running services, and the best way to achieve this is through virtualization, offering only a fraction of the node resources. Since physical resources are placed all across the globe, a distributed virtualization platform is needed. Multiple VM's are treated as a single one, introducing a new concept known as slices.
- Unbundled Management: PlanetLab puts almost at the same level the services running by users and the infrastructure services, such as slice creation, software monitoring, performance monitoring, etc. Being a continuously evolving platform, they want the infrastructure services to evolve not depending on the bundled platform, parallel to the services deployed by its users. This will allow different versions of the same infrastructure services to be compared as the platform evolves. Note that infrastructure services will have more privileges than normal services.
- Chain of Responsibility: Nodes are geographically distributed across the globe being managed by different autonomous organizations. PlanetLab Consortium (PLC) is the entity responsible for the operational stability of the whole infrastructure. It acts like an intermediary entity between different organizations (that manages their nodes individually) while the system itself knows which organization is responsible if something goes wrong. One user does not have to trust many different organizations to deploy their services, since PlanetLab will act as their intermediary, granting more freedom to the whole system. This is enforced through trust relationships between node owners, authorities and slices users, which will be explained later on.
- Decentralized Control: Each organization maintains and is responsible by the resources it provides to the PlanetLab infrastructure.
- Efficient Resource Sharing: A fair distribution of resource among slices is maintained. Slice and resource allocation are decoupled, meaning that each slice must assign resources. Each slice guarantees "best effort" behavior of its resources, without starving other slices.

Actors and Trust Relationships The following describes the previous referred to chain of responsibility. The main actors in the PlanetLab infrastructure are:

- Owners have the upmost control of their nodes, but delegate management to PLC. Owners trust PLC to provide with software which manages the network, and to provide resources latter available to users. PLC trusts owners to keep nodes physically operational and secure. Optionally, owners can control resource allocation through policies.

- Users are the ones who create their slices (through mechanisms provided by PLC), to run their services. Users trust PLC to create slices in which they are able to run their services, while PLC expresses their trust with access credentials to PlanetLab.
- PlanetLab is the trusted intermediary, which manages the nodes on behalf of its owners, and creates slices for their users. PLC is divided into two different actors. A Management Authority (MA) is responsible for the monitoring of all nodes, and a Slice Authority (SA) keeps track of all slices and their mapping to users. While the MA trusts SA to map its slices to users, SA entrusts MA to provide working Virtual Machines (VMs) and user control.

The core element of the PlanetLab architecture is the slice. A slice consists of a set of allocated resources distributed across all nodes. Nodes are assigned to slices, consisting of virtual machines providing computational power, storage, network resources, in a controlled manner. Many virtual machines can coexist on the same PlanetLab node, running on top of a monitor, called Virtual Machine Monitor (VMM). This entity is responsible for the auditing of each virtual machine system activity so that the resources are distributed fairly to avoid starvation. Remember that since PlanetLab is seen as overlay network, services must coexist with each other, being that fair distribution of resources is an important requirement. Virtualization in PlanetLab is achieved through the well-known Linux vserver. Unlike commercial products which can provide 10 VMs per machine, or a common runtime like the JVM or Microsoft Common Language Runtime (CLR), in PlanetLab a much larger number of VMs may run on the same node, requiring a more lightweight mechanism like linux vserver.

A Node Manager (NM) (holding the VMM), runs on each node and is responsible for its resources management and association with slices. The NM maintains resource pools, namely RSpec, which basically comprise the resource allocated by the node. The SA creates the slice, which is stored on its database. This authority is responsible for maintaining the slice association of its users as well as resource pools. Slices will subsequently be instantiated, and resources assigned to each node. This can be done either “manually” or by an infrastructure service called Slice Creation Service (SCS). The Auditing Service (AS), which is also an infrastructure service running on the node, is responsible for watching node failures or aberrant behaviour. A MA is responsible for keeping this software updated (NM, AS, SCS), maintains a database of its registered nodes, and also takes action if a node is not behaving appropriately. Other infrastructure services are also deployed, enabling resource brokerage (simplifying resource binding), node health monitoring, and discovery services for resource availability.

OneLab

Initially PLC was the only intermediary, but was later decoupled into two different parts (MA and SA). This implies a great improvement of PlanetLab scalability, as different parts of the PlanetLab infrastructure may evolve independently. Currently PLC is the only management authority; federation is achieved when more groups start to play their role. OneLab[48] project aims to enhance AS as a federated system, by extending it with wireless technologies through the federation of wireless testbeds. This provides users with the use of a whole new set of wireless technologies offering a greater variety and complexity of services. The main purpose of OneLab is to provide researchers with one network testbed, by uniting each institution network resources. It is a European founded project that began in AS Europe, being already federated with PlanetLab Japan and Korean Private PlanetLab.

To support this kind of network ecosystem, some challenges were found in the PlanetLab architecture, which are currently being studied[49]. Node information maintained by PLC must be more precise, since some experiments will require node topology dependency; Furthermore with wireless technologies, concurrent experiments introduce a problem due to wireless spectrum sharing issues. It is also not envisioned on current PlanetLab architecture that some specific resources might be only available on a specific slice, such as a UMTS network interface.

2.3.3 Wireless EXperimentation Tool

Wireless Experimentation Tool (Wextool)[50] is a tool developed in Institut National de Recherche en Informatique et en Automatique (INRIA), addressing the problems of experiment creation in wireless network testbeds. Developed for use in any computer, wextool can be configured in any wireless-enabled station/computer. A workflow [51] is here addressed to provide easy experimentation on wireless testbeds. Experiments are first processed by issuing a scenario. This scenario is an XML file that will contain nodes layout definitions, and the definition of the whole environment, by specifying software components and configurations to be made. Furthermore, traffic characterization and data to be captured in the experiment are specified in detail, along with the number of times the experiment will run. This scenario file can be created either manually or using a wextool GUI. Once this scenario is established, the experiment is performed, packet traces being acquired and sent to the server. Optionally, real-time monitoring can be enabled to observe traffic-load and packet loss variations, and other metrics. When the experiment is finished, processing of data takes place, which will synchronize and gather packets traces to a central database for further analysis. Wextool also provides ways to analyze data, allowing the correlation of different wireless metrics. This tool also lacks integration with applications for user-generated metrics.

2.3.4 Network Experiment Programming Interface

The three approaches in network testing, have many different advantages. Simulation has the most notable repeatability and configurability, being able to change all variables on a network, from the physical layer to the whole network model. With the emulation approach, virtualization allows an experimenter to emulate a larger set of resources, with much less hardware. Finally, testbeds represent the real network, providing the reliability that the emulation and simulation approaches cannot easily provide.

Researchers at INRIA, believe that mixing all tools can greatly improve a network researcher productivity, and bring them more freedom when trying to create complex experiments. Lack of resources, hardware issues, complex network topologies, reproducibility issues are some of problems encountered when trying to create more complex experiments. In such situations, this mix of environments can be very useful. For example, when testing a video streaming application using 20 nodes where only 10 exist, NS-3 could simulate the remainder. This is one example of the flexibility between environments they wish to demonstrate.

Architecture

For this purpose, they created the Network Experimentation Programming Interface (NEPI) framework[52] which tries to interoperate all the experimentation tools, creating what they call “mixed experiments”. It allows a researcher to use NS-3, PlanetLab, Netbed, OMF-enabled testbed, or any other infrastructure, and switching among them easily through NEPI API. Their aim is to provide a uniform manner to simplify the typical experimentation tasks such as measurement collection, network topology, across all different environments.

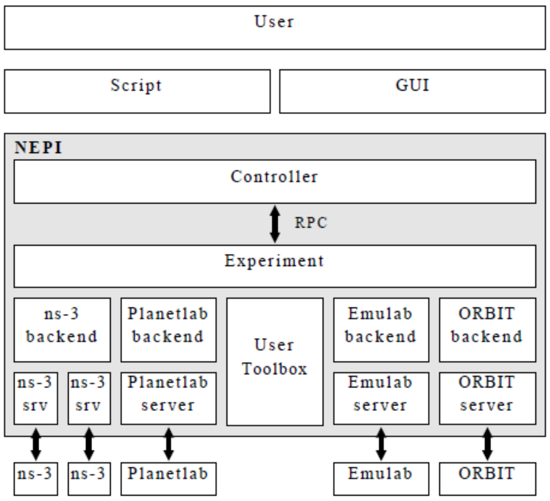


Figure 2.13: NEPI Architecture

The NEPI API is accessed through a Controller which is in charge to interact to a remote daemon responsible for running the experiments. The controller is where user authorizations take place, as well as user assignment for each experiment, through remote procedures calls. The Experiment is the one holding all the experiments description. They keep track of experiment progress and related data such as results, and are responsible for the deployment and monitoring of every host. NEPI functionality resides mostly in an Object Model Representation, to describe experiments. Backend and Server are the entities providing integration with experiment environments. The Object Model is extended through Backend implementation where Server holds the environment, which can be a NS-3 simulator process, OMF Aggregate Manager, Netbed `masterhost` or any other experimentation support resource. The user toolbox represents a higher abstraction of each Backend objects, providing access to all Environment Backend Objects.

The Object Model

Creating experiments using the NEPI object model follows the same method as creating an integrated circuits layout. Modules with certain properties can connect with other modules of the same type, or create composite modules, to describe a more hierarchical representation. NEPI uses six types of entities: Object, Connector, Attribute, Trace, Result and Operation:

- Object represents a functional unit like a PlanetLab Gateway, an Emulab Node, an

NS-3 Object or an OMF resource;

- Connector serves as a bridge between objects, for instance, an NS-3 IPv4 Object.
- Attribute are properties of an object (e.g. Transmit Power, Link Speed, Source, Destination, etc);
- Trace represents a trace of an event occurring on an Object;
- Results describes something generated by an Object;
- Operation is an action scheduled on an Object (e.g. olsr-daemon to execute in a Emulab Node).

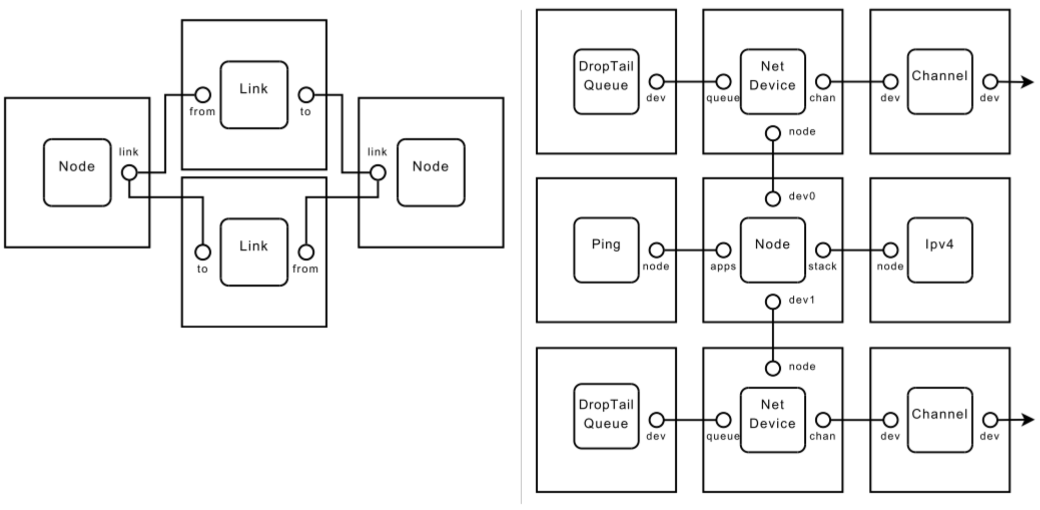


Figure 2.14: ModelNet (left) and NS-3 (left) object diagram examples

ModelNet emulator topologies are written in an XML containing edge and core nodes of a network. Figure 2.14 describes one way to model ModelNet and a Ping example application with NS-3 Object Model, using NEPI: One Node object has one Link object, with the source (from), and destination (to), attributes; the Ping object is connected to a Node with an IPv4 address. Also described are the two wireless network interfaces belonging to a Node. Although this is a very low-level example, one way to reduce the level of complexity of the model is to aggregate all Node related objects (except Ping), to only one object.

NEPI Framework brings a new approach to network testing, aiming to create a heterogeneous network testing environment. Currently it is still in prototype stage, and only the NS-3 backend has been implemented. Controller capabilities are yet unknown, so yet there is no sign of methods to control network resources or as to monitor them. Supporting NEPI framework, a front-end called Network Experimentation Frontend (NEF) has also been created, but no documentation supporting this frontend has been written.

2.3.5 Summary

These testbeds vary from a simple installation of operating system in nodes, to custom management software. UMIC introduces PXE for software deployment, while using SNMP for its monitoring. APE testbed created a way to describe mobility through scenarios files, and it uses mackill tool to create topologies. A software package facilitates the deployment on testbed nodes. MiNT mobile testbed is distinctive for embedding NS-2 in nodes, and for tracking movement using a RFID location system. It is a relevant initiative showing a cost viable manner to create a mobile testbed, supported with a platform that allows users to easily create their experiments. The University of California, Santa Barbara (UCSB) testbed introduces out-band and in-band management by separating control network from experimentation network, although no further management is established, apart from tools to analyze and manage the wireless network. ORBIT currently is one of the biggest deployed wireless testbeds. They have created a tool to optimize its management. Currently this tool is used in many testbeds deployments around the world. Nitlab is one of such examples but they focus more in scheduling mechanisms to increase the number of experiments to be executed simultaneously. They have developed a wireless spectrum slicing mechanism that separates the channels one user is allowed to use. AMazING testbed is another example of an OMF deployment, and their objectives are to provide a testbed for the study of mobility scenarios. By integrating NS-3 in their deployment, they aim to bring the controllability of such simulation environment in live experimentation, similar to what MiNT have done. They've also extended OMF to facilitate results collection, giving support to a wide number of metrics. The EXTREME testbed is similar to OMF, aiming to create a framework to manage a testbed while supporting multiple wireless interfaces. Although, they fail to integrate applications to generate their measurements. Their infrastructure does not allow this kind of integration and is only reduced to network metrics. Although compared to ORBIT testbed, it has a larger support in terms of wireless networks. Also similar to OMF, wextool aims to provide a framework for users to define their experiments, although with limited methods to deploy software and collect metrics. PlanetLab is a planetary testbed distributed across hundreds of sites, delivering a platform where users are allowed to deploy their services. Its infrastructure is optimized for multiple services to co-exist.

Emulation gathers the best of both worlds. With much less hardware, it allows the reproduction of larger scenarios, through virtualization. While deployment costs are lower, it also leverages results reliability by running on real hardware. The most notable example is, the Netbed where "testbeds are deployed as clusters". They are provisioned with virtualization capabilities on top of this hardware and a whole infrastructure, which includes collaboration, monitoring, deployment, and experiments creation through NS-2 scripts. The workgroup behind this platform were the creators of Frisbee, which was designed to restore nodes disk and to facilitate the software deployment.

Almost all of these testbeds share similar problems: measurements are reduced to the analysis of log files; software deployment and execution of the experiments are usually done manually. Furthermore, the whole deployment of the testbed is expensive and its maintenance incurs additional costs. Testbeds are made for specific purposes or project objectives, and are subsequently useless. This leads to the lack of testbed reutilization.

NEPI Framework aims to bring an programming interface to unify all the different environments in network testing, mixing the use of network simulators, OMF-enabled testbeds, Netbed and PlanetLab. This is a great initiative, although it is still in prototype stage and only network simulators backend is supported.

The concept of testbed management system has arisen, as testbeds require more instrumentation capabilities, and to ease management actions to overcome the problems of maintaining and deploying a testbed. DES-TBMS is distinctive and provides this kind of system for a hybrid testbed, with an interface to manage experiments, and tools to monitor and analyze results. OMF is also notable, being a current state-of-the-art tool to manage and easily create experiments. It features a more dynamic and scalable method than DES-Testbed, which lacks easy software deployment, and a less scalable way to create experiments (DES-Cript in XML) when compared to a language designed and integrated with the whole platform OMF. OMF Workgroup is making its way towards the creation of a portal that makes accessible the testbed to users, although and according to their future planning, the platform is more designed to schedule experiments and collect measurements. Users still need a more in-depth knowledge of the whole framework, since the process is still complex and decoupled.

Chapter 3

Testbed Management Platforms

Previously different approaches on network testing were shown: simulation, emulation and testbeds deployment. When using a simulation test, experiments are granted with a completely controllable environment, with all variables of a network at his disposal. Usually coming under the form of discrete event simulators, OMNeT++ and NS-3 are the most resounding names among the network community. A network is modeled by the tester and then simulated by software. There are a variety of network scenarios, and simulation alone cannot provide the reliability of results. Nonetheless simulation proves very important in preliminary tests especially when compared with theoretical values.

Due to this lack of reliability, network test community comes to lean on the deployment of test facilities. Testbeds arose as a solution to this problem, as well as software residing on these, suggesting new approaches, and different objectives.

Previously, the AMazING testbed owners had been confronted with these problems [53] and aimed to solve them by installing and developing a management infrastructure on their testbed. The choice lead to OMF, which is an active project aimed at solving these problems. It is a state-of-art tool, which provides easy software installation and controllability for either a testbed owner or an experimenter. In this chapter a more analysis of this tool is made along with the testbed deployment that will support this work.

3.1 cOntrol and Managment Framework (OMF)

The OMF[54] framework is a software package providing tools for control, management and measurement of testbed resources. While a testbed owner gains access to an effective management of all resources, the users can easily instrument their experiments, run and collect all their results in a centralized way. Instrument here, means to configure all network parameters, execute applications, results collection, where without OMF these tasks would slow the whole testing process.

OMF was designed to achieve reproducibility, and greater repeatability on experimentation. Its main design principle centers on the experiment description. This description will contain all the configuration and information of resources, and then delegate it to OMF, where the resources will be deployed and configured, experiment executed and finally results collected (Figure 3.1) From a user perspective, the basic workflow submitted is based on three steps:

1. The user is asked for a script, called experiment definition, which will contain applications, OS images, topologies, a specific node behavior or network configurations;

2. OMF will deploy and configure according to this experiment definition. When all resources are configured, the experiment is executed and measurement data is collected;

3. Finally, the measurement data is stored in a standardized database format, and can be used for later analysis.

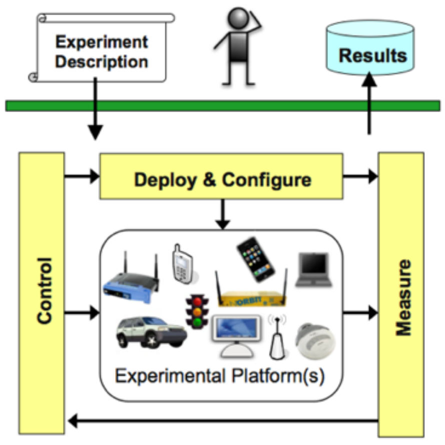


Figure 3.1: OMF Functional Principle

3.1.1 Architecture

The OMF architecture, related to the 5.3 version, is structured as illustrated in Figure 3.2. Three main entities maintain the control of the testbed. The Experiment Controller (EC) is the entity that the users get access to. Just described in the previous workflow, an experiment definition is passed down to this entity to be executed on Testbed. There it will request Aggregate Manager (AM) for resources initialization, loading disk images if necessary and communicating with the Resource Controller (RC) that each node holds, to issue network configuration commands (e.g. essid, ip) or applications initialization. When all resources are up and installed, the experiment starts. While an experiment is running, the EC may ask for nodes to start their applications or modify their experiment/application parameters. Nodes respond through the RC module. When there are applications still running, results will be periodically collected and sent to the AM and gathered in a Database, which will be later accessed by AM services. A correct deployment of the OMF should have a control network and the experiment network. Control traffic should be separated, or results may be inconsistent, due to the background traffic created by the control entities. These modules will be explained in the next sections, from their architecture to the whole software stack.

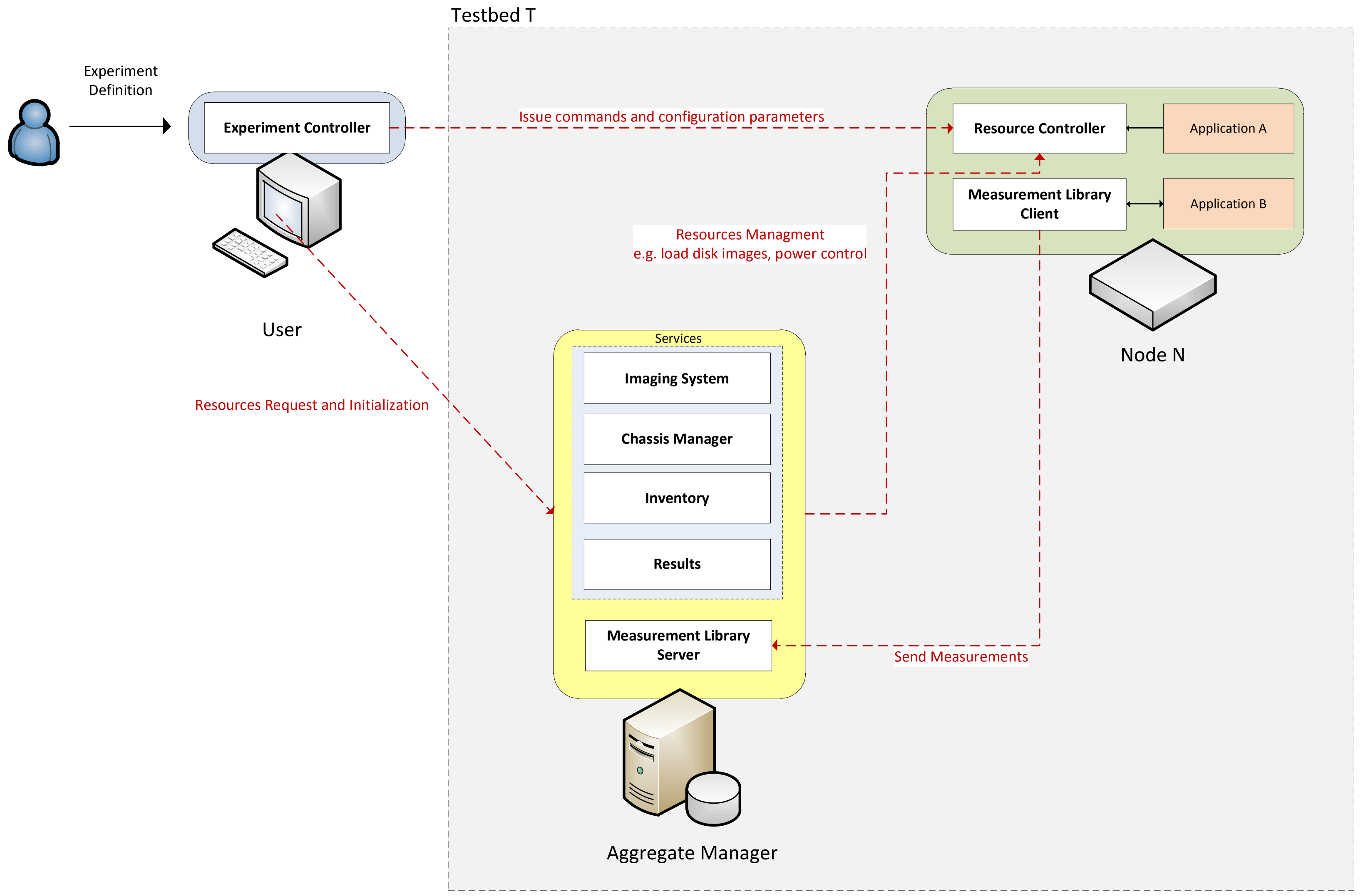


Figure 3.2: OMF Architecture Diagram

3.1.2 Aggregate Manager

This module also called as Grid Services, is the one responsible for the management of all resources within a testbed. It consists on a collection of services that provides to an OMF enabled testbed the following capabilities:

- An imaging system that let experimenters load/save their own images to the nodes;
- An inventory which maintains information related to all resources, such as drivers, hardware, location of node;
- A chassis manager which control the power of resources;
- Boot management of resources and access to the measurement databases through web services.

The Aggregate Manager is responsible for holding all these services described above, as well as the Publish-Subscribe Server, later explained. OMF created a kind of ruby meta-description, to describe each service heterogeneously, increasing the extensibility factor of the AM. This meta-description is provided by the Service Core API . A service must be placed in the correct package in the form of `ogs_<service_name>` and each service must be described using `s_name`, `s_param`, `s_description` keywords. All classes in the packages using that naming form will be loaded. After that, ServiceMounter will mount each service and start the HTTP Server that handles all requests. For services that needs to run a background process (e.g. Frisbee), the AM provides an AbstractDaemon API to extend. Figure 3.3 illustrates the AM software package.

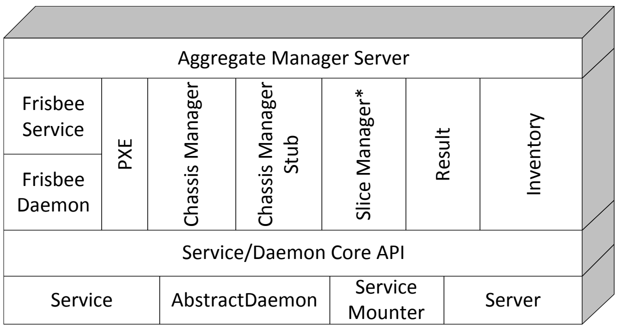


Figure 3.3: Aggregate Manager software stack

Disk Imaging System

Aggregate Manager allows users to load disk images on nodes to facilitate the software deployment. This is done using a "pre-boot" system (PXE) together with an application (Frisbee) that will distribute the image to nodes. It will act as the streaming server, and the clients just need to receive the chunks of the disk image.

The Imaging System consists of a Frisbee daemon, a PXE server and one or multiple clients of both of these services. PXE is an execution environment whose objective is to boot up nodes, using a network interface and independently of operating systems or data storage that is already on disk. It is deployed with a server and client having both Trivial File Transfer Protocol (TFTP) and Dynamic Host Configuration Protocol (DHCP) services running. The client tries to locate information about the existent AMservers (through DHCP). Once the client finds the appropriate server, a Network Boot Program (NBP) is requested and transferred via TFTP. In the case of OMF, this NBP will be the Frisbee. When transferred, the NBP is stored on Random Access Memory (RAM) and then executed, where it will begin the image transfer. The image is sent to one or multiple nodes over a reliable multicast session and booted afterwards. The image transfer is handled by Frisbee, while the boot management is made by PXE. In the case of failure, a default image is booted instead. Frisbee can distribute the same image to multiple nodes simultaneously, using multicast. Image transfer is made through a reliable multicast protocol to ensure packet delivery. Note that, Frisbee was initially created to support the existent "testbed clusters" at Netbed, where an image is loaded to dozens of nodes at the same time.

Inventory

The AM also stores hardware and nodes information, in a database, which is called the Inventory. This inventory contains a set of tables, which maintains information about motherboards, node locations, devices available and PXE images information.

Chassis Manager

The Chassis Manager (CM) is another service provided by the AM, responsible for the power management of nodes. It comes with a core implementation that depends on specific hardware with power controllability. A stub is also available in AM services, which provide custom power management to other types of resources. This stub comes with no implementation, being the task of testbed owners to extent it at their own will. The AMazING testbed makes use of this capability, enabling power management through a Router. subsection 3.2.2 refers to this implementation.

3.1.3 Experiment Controller

Also known as Node Handler (NH), the EC role is to allow users to execute their experiments. The user comes in contact with the EC to execute commands such as the load of an image to a node(s), or execute an experiment. The EC cooperates with the AM only when the initialization of resources takes place i.e. the disk images load. When resources are configured, the EC will communicate with the nodes sending commands according to their Experiment Definition.

The the EC is bundled as an application to be installed in the users terminal. It holds responsibility for configuration, deployment and experiment execution tasks. Usually these tasks are sluggish and require manual input each time tests are made, but using the OMF all of these items are controlled through the language created for this purprose. It is the frontend of an OMF-enabled testbed, being called the OMF Experiment Definition Language (OEDL). This language is part of the EC, being the major module on the Experiment Controller. We will talk about this language and give an overview of the EC software architecture.

OMF Experiment Description Language

OEDL is a Domain Specific Language (DSL) that enables a user to instrument his experiment, making all usual configuration and deployment tasks more simple and automated as possible. Through this language, the user will be able to describe his experiment in a higher abstraction, being the logic centered on the script. This approach allows a user to have high controllability while providing the automation in execution. In short, writing an experiment implies the creation of an OEDL script, called the Experiment Definition. Its creation, involves three different "phases":

1. Resource Initialization and Configuration – the user will have to configure their resources, defining their network parameters, disk images do load, applications running and measures to capture.
2. Events and Tasks Definition – The flow of the experiment is dynamic, meaning there is no lifetime specification in the description. OEDL provides a set of events that are triggered during the experiment, either the user can creation its own. For instance, when resources are all configured and ready to run, an event is triggered meaning your experiment will start to run. Experiment tasks must be defined within the scope of this event handler.
3. Graph Integration (Optional) – The user can see their measurement data in a web page, by specifying the graphs to generate, in the experiment definition.

Resource Configuration

OEDL syntax is Ruby[55], but being a Domain Specific Language it was designed for people with general programming skills to easily create their scripts. Users will semantically describe their sequence of actions to execute on the testbed, instead of issuing terminal commands to nodes. Users only have to learn these semantics, to create the experiments. For instance, if an experimenter wants to configure one node network parameters of a wireless interface, the usual approach would be to connect manually to each node issuing commands such as:

```
$ iwconfig wlan0 essid test mode managed
$ ifconfig wlan0 192.168.1.1 netmask 255.255.255.0
```

Using OEDL semantics, instead you will define a group containing the node and simply change the IP address of the desired interface, such as the following: Highlighted in red are the most important parts:

1. The definition the group, with a name, a set of nodes associated and the desired properties (IP, Extended Service Set ID (ESSID)). The node name follows a resource naming convention, to enforce their uniqueness if multiple OMF testbeds were coexisting.
2. Node properties are defined through a resource path, having multiple properties for each type of network interface. `net.w0.ip` refers to a network parameter (`net`), of the Wireless Interface 0(`w0`, Ethernet would be `e0`), being the ip the property to configure.

Example 1 OEDL to configure network parameters

```
defGroup("GroupName", "omf.testbed.node1") do |node|
  node.net.w0.ip = "192.168.1.1"
  node.net.w0.netmask = "255.255.255.0"
  node.net.w0.essid = "test"
  node.net.w0.mode = "managed"
end
```

Other wireless related properties such as bitrate, channel and transmit power can also be changed. In OMF this kind of expressions are denominated as Resource Paths.

Applications

Normally a testbed is used to test a prototype of some piece of software, which can be a routing protocol, a kernel module, an application, etc. The common way to test applications in a testbed is by remotely connect to each node and starting them manually. The main drawback of this approach is that it requires too much user input, and when some synchronization between applications is required, the experiment success is dependent on the user quickness or his sense of timing. Additionally, analyzing results implies filtering all the log files to gather the relevant metrics. To solve this problem, OEDL lets users instrument their applications, with ways to generate metrics centrally. Defining the instrumentation implies doing it only once, with no input required from the user. We will have to create an Application Definition with similar semantics to the example above. This application definition is an interface, which is translated to a command that will be executed on the desired group. For the measurements, if your application is OMF Measurement Library (OML)-enabled it only has to declare which metrics will be injected to the results. More on this subject in subsection 3.1.7.

Topology

Topologies provide users with a way to layout the network in an experiment. OEDL allows the user to create his own network topologies, creating nodes and defining links between them with different properties. This is useful to test multi-hop routing protocols, limiting the “visibility” of the nodes. In Ethernet interfaces, a user can define different routes and apply packet filters. In Wireless Interfaces, these topologies are used to enforce link on the interface. This is done with tools such as *iptable*[56], *ebtable*[57] or using Netem. OEDL makes this configuration process transparent to users, providing semantics to define these topologies.

Disk Image Loading

As explained previously, an OMF enabled testbed can install entire disk images onto nodes, easing the dependency management necessary to deploy applications. OEDL Language provides expressions in its grammar to enable disk imaging. The EC maintains a repository where it resides application definitions and special scripts which enables part of the OEDL functionality. When the user asks for an image to be loaded, a special script in this repository will be invoked, to configure nodes and trigger the image load process. In current versions of OMF image loading and experiment execution consists in two different processes, although both are described in OEDL scripts.

Stack

The EC stack is defined mainly by the implementation of the OEDL Environment, as illustrated in Figure 3.4. A set of packages powers the OEDL implementation, having the NodeHandler as an entry point of the EC Command-Line Interface (CLI). This NH will load the experiment and store all experiment related information. The implementation of the OEDL language makes heavy usage of Singleton objects to store the experiment state and communications with the nodes. Sucessive runs of experiments, could corrupt information contained in these Singleton objects. In addition, this implementation does not provide libraries to access the information contained in the experiment definitions.

During the experiment execution, the EC will launch a web server that allows the experimenter to see logs, measurements and scripts. Although, it is a very limited web application meant to monitor the experiment.

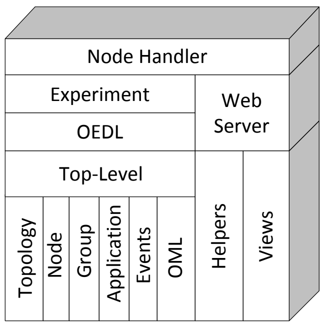


Figure 3.4: Experiment Controller software stack

3.1.4 Resource Controller

This module represents the controller entity of every node available in the testbed. His main purpose is to respond to commands issued by the EC. For example, when a group defined in the OEDL script sets a resource path, the EC will send the resource path data and value to the corresponding RC which will handle and translate to the appropriate command that will configure the network interface. All the communication uses the Extensible Messaging and Presence Protocol (XMPP), whereas the communication model between OMF entities is defined by the OMF Protocol. In current versions, Resource Controller just responds to the commands issued by EC, taking no initiative in the whole experiment process.

3.1.5 OMF Protocol

In early versions of the OMF, communication between the different entities were carried out using a simple text protocol through a TCP/multicast server. Aside from implementation limitations, in a future environment where different testbeds are connected together (federation) this approach faced some problems. For instance, resources are not guaranteed to be unique in a federated environment, and multicast traffic might not be able to cross network boundaries. Furthermore, the resources available in a testbed, did not predict different types of resources or relations between them. As the project became larger, the objectives grew beyond the need to manage ORBIT testbed, therefore changing their design.

The entire experiment communication infrastructure was refactored, thus introducing the Extensible Messaging and Presence Protocol[58]. It consists on an Internet Engineering Task Force (IETF) standardized XML protocol which is mostly used in presence, messaging and real-time applications. Its foundations are supported by a client/server architecture, in which all peers talk with each other through a XMPP server who will be the abstraction layer between all XMPP communications. The server is responsible for the connections between different entities and maintains the appropriate information among peers such as presence state, sessions, contacts, subscriptions. Servers connect to each other when different domains are involved.

One of XMPP advantages is its extensibility. Through XML, one can easily extend the XMPP core protocol[59] creating the so-called XMPP Extension Protocol (XEP). XMPP usage is flexible, so organizations can maintain their server implementation and create their own private extensions. The core protocol provides the base XML communication layer, authorization and authentication mechanisms, presence availability and contact lists. Among the dozens of extensions available[60], one can highlight XMPP over Hypertext Transfer Protocol (HTTP) and Publish-Subscribe, as the ones of most interest to OMF.

On top of XMPP, OMF created a custom protocol dedicated to the orchestration of an experiment. This protocol, denominated as the OMF Protocol, is the one that enables to resource configuration, management of experiment resources and application execution. Resources are organized in a hierarchical structure to distribute resources between experiments and across different testbeds. They used the Publish-Subscribe XMPP Extension, where different content publishers can ”asynchronously” notify content to its subscribers.

3.1.6 Publish-Subscribe Scheme

All communication between resources and experimenters are made through XMPP using Publish and Subscribe extension[61]. All active entities interaction are best described in a tree diagram (Figure 3.5): “OMF” represent a testbed that is formed by ”System” resources that will be distributed across “Slices”. Each resource listens for actions under the System group. The gray nodes are created by default when deploying OMF, while the remaining ones are created by EC when executing an experiment. Colors imply different roles in this communication model. When a user wants to execute an experiment, they request a “Slice” with “Resources” to use. As described before, when an OEDL script is loaded, groups will be created to make use of the testbed resources. Each group will also have a corresponding node. AM subscribes to the all system nodes he manages to issue low-level commands. RC will subscribe to its OEDL group and assigned resources, where it will receive configuration commands. EC subscribes to all the nodes it creates to acknowledge group establishment, and to monitor nodes state.

Currently the EC is the one responsible for creating the slice and its resources, although this is envisioned to be created by third-party tools with the intent of using an OMF testbed.

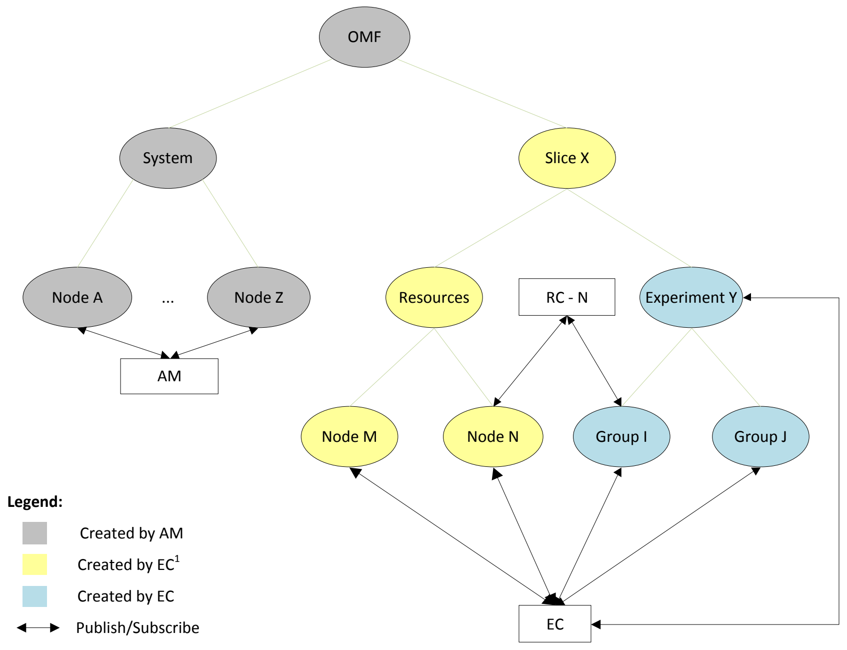


Figure 3.5: PubSub Nodes Tree

The Publish-Subscribe is currently being refactored to be used in conjunction with PlanetLab. This migration began in version 5.3 and it will be finished in version 5.4.

3.1.7 Measurement Collection

Gathering measurement data is one of the key challenges when working with experiments on large-scale testbeds. Often data comes in some form of log files, whose formats can vary. All the log files must be collected from all resources and passed through some sort of serialization in order to extract the most relevant data. This sluggish process of measurement collection adds overhead in the whole experimentation workflow, and thus slows the whole experiment objective, focusing less on the application or routing protocol to test.

The OMF Workgroup comes with a measurement collection framework to solve these problems, namely the OML[62]. This framework provides means for an experimenter to define the measurements points, processing of the results, enabling him to organize his data into a single database, avoiding the fragmentation of having such numerous log files among all the resources. OML efforts deliver a "generic" and simple framework that is able to measure anything instead of being restricted to network metrics. The principle behind this framework consists on streaming measurement data from clients to a server responsible for the data aggregation to a single database. Experimenters can easily instrument their applications with various methods through the OML client libraries.

The server only task is to receive data from an OML-integrated application and join all the data in a database for a given experiment ID. In addition, OML framework also provides a proxy server. This proxy server is responsible for locally storing the measurements during experiment that will be sent later to the OML Server. It consists of a queue of measurements to be sent to the collection server, in which the dispatch is activated through

Pause/Resume commands from applications. The usage of this proxy is made by telling in the experiment definition if the node is disconnected. Disconnected nodes are more suitable when network restrictions are present[63]. Usually when experimenting with mobile nodes, control network may be unreachable or out of range. Depending on the testbed configuration it might not be viable to send the measurement streams through the experiment network, since it will introduce traffic overhead and thus inflicting error on the results. Furthermore, when doing experiments that involve high traffic rates, sending the measurements could congest the network, causing measurement data loss. The solution is to buffer the measurements temporarily (when the network is unavailable or at the end of an experiment) through this proxy server.

Measurement Library Client

Doing experiments correctly, involves integration of applications with OML. Its main advantages are the previously explained results centralization and a power filtering mechanism that comes with OML. It can be done by either changing the application source code or by defining a wrapper on top of the application. OML libraries were made with simple integration in mind. If your application is integrated with OML, the resource controller will take care of the rest, and the user gains access to filtering mechanisms that enables users with metrics data control. A filter consists of injecting data to the database that will pass through series of arithmetic calculations (average, sum, etc). For instance, instead of injecting all the caputed samples, an average of a smaller number of samples could be injected to the database. Users decide when this process happens by specifying a sampling/interval rate and the filters to apply..

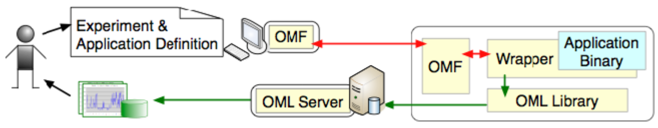


Figure 3.6: OML Wrapper Application

Defining a wrapper (Figure 3.6) is the easiest way of integrating an application, specially when the user only has access to the application binary. This is done using the OML Ruby client library. Using a wrapper means that the application will have logs or output, that needs to be parsed in order to inject the metrics to the database. To create the wrapper, one must, first, define his measurement points and associated metrics, following by the application initialization and finally process the output from the application where the injection of metrics takes place. A Measurement Point (MP) is here referred as a set of metrics. These metrics are expressed in tuples, which contains the name of what to be measured and the type. MPs are then translated to a “schema” that is attached with every measurement stream to OML server, representing the format which the metrics must injected. The measurement streams are sent during experiment, and are part of the text protocol that OML uses to transmit the results between clients and server. The protocol consists on sending measurement streams, which will contain information about the OML application, the experiment, the MP structure information (the schema) and finally the results.

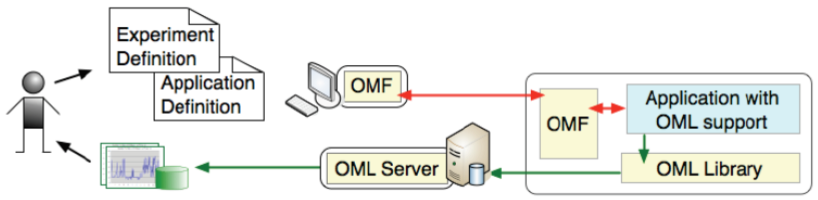


Figure 3.7: OML Native Application

Besides the wrapper, the user can make a more native integration within applications with OML (Figure 3.7). In the wrapper creation, the user is limited to output/logs thrown by the application, while in this native approach the user get access to the application source code being able to inject more metrics. The main drawback of a native integration is the increased difficulty of integration when dealing with third-party applications such as mgen or iperf. However, we gain access to much more information and thus able to inject much more data to the results. For example defining a wrapper ping will allow the user to get metrics such as Time To Leave (TTL) and Round Trip Time (RTT) that we gather from ping output. A more native integration of the ping would give access to the Internet Control Message Protocol (ICMP) headers of each packet. Since the user can choose the metrics to use in the experiment, the native approach will provide a higher variety of results.

OMF and OML are completely independent. Although, some of the OMF components provide seamless integration with OML when both are present of the same machine. In such cases, the database results are automatically created in the filesystem without the need of any input. In addition, AM provides web services related to experiment results.

Some well-known applications such as iperf are instrumented with OML. They also create a traffic generator and a traffic sink, which can be of use to user experiments. The OMF is a testbed management framework, which manages a testbed while providing with controllability of experiments. OML introduces a generic framework for metrics generation, which allows one to capture any metrics that an experimenter wants to, without being restricted to network metrics. The experiment is not subdued to a flood of log files along with its entire parsing.

3.1.8 Portal

A Portal[64] was developed to support users with manageability of their experiments on the National Information and Communications Technology Australia Ltd (NICTA) deployment. This platform was built on top of Redmine[65], and is only dedicated for management of reservations, experiment related scripts and artifacts. Users will have GIT[66] repositories to manage their experiment definitions, while they are able to reserve of testbed resources. It offers three different types of reservations: ideal topologies, time slot and as soon as topology nodes are available. Ideal topologies happen when topologies contained in experiments meet certain conditions (e.g. RSSI $< 20dbm$, PLR $> 5\%$). Time-slot refers to an interval that is used to reserve testbed nodes. Besides these reservation mechanisms, the portal also provides auto generated graphics of the results. In the future, they aim to enhance results processing by providing support for a statistical environment, namely the R language[67]. However, their portal is limited, being only a tool for users to manage scripts version and reserve testbed resources.

3.2 Advanced Mobile Wireless Network Playground

The AMazING[68] consists of a wireless testbed, composed of twenty-four nodes located at the Instituto de Telecomunicações - Aveiro. As explained previously, simulations sometimes fail to provide reliable results due to incorrect validation of the network models. In addition, in previous deployments such as the Daidalos project[69], problems were found that were later solved in the AMazING deployment. In the Daidalos testbed, the authors were to make use of New Generation Networks (WiMAX, Wifi, and UMTS) within a federated environment. The purpose of the Daidalos project was to study the foundations of an All-IP operator network, for future support of a broad range of personal services. These problems were on the monitoring and maintainability of the developed software components, nodes operating systems and testbed uptime. AMazING testbed operators stated important key requirements that address these problems, which distinguish this testbed deployment for the one previously described. The main requirement was the need of a reproducible environment for users. There were several considerations in the testbed creation:

- Low cost/power: Since there are a relative larger number of nodes, these should have a low cost and power too. This reduces maintainability costs of the testbed, as the cost of creation.
- Reduced Radio Interference: The testbed was setup at IT rooftop, to reduce interference elements, such as people movement.
- Wireless Radio Technologies: This testbed aims to serve the wireless technologies studies. A set of wireless radio interfaces must be supported in order to support these studies.
- Processing Power: Nodes must have the processing power and storage mechanisms to run bleeding-edge software and to analyze results.

In addition, as the project was evolving, the need for management platform rose, introducing a whole set of software components to be deployed within these requirements. They used OMF as the main tool with additional extensions and improvements.

3.2.1 Deployment

The testbed was setup an Instituto de Telecomunicações - Aveiro rooftop; consisting of twenty-four nodes distributed in a $1200m^2$, each node being separated by eight meter from each other. Due to the requirements listed previously, hardware was deployed as a low cost Comercial Off-The-Shelf (CoTS) device, i.e. a low cost hardware package fitting the node testbed needs. The nodes are protected with enclosures since they are exposed to environmental conditions therefore needing a watertight, Ultraviolet (UV) and high temperatures resistant protection. All node component positions were taken in consideration, to be sustainable under aggressive weather conditions. These include board position, enclosures material to avoid excessive heat and water condensation. Figure 3.8 illustrates the final product.

Hardware specifications (see Table 3.1), include radio support for Wi-fi N draft standard and affordable processing power, for users to deploy more recent operating systems (e.g.

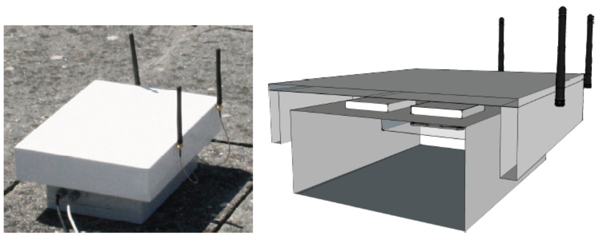


Figure 3.8: Node Chassis

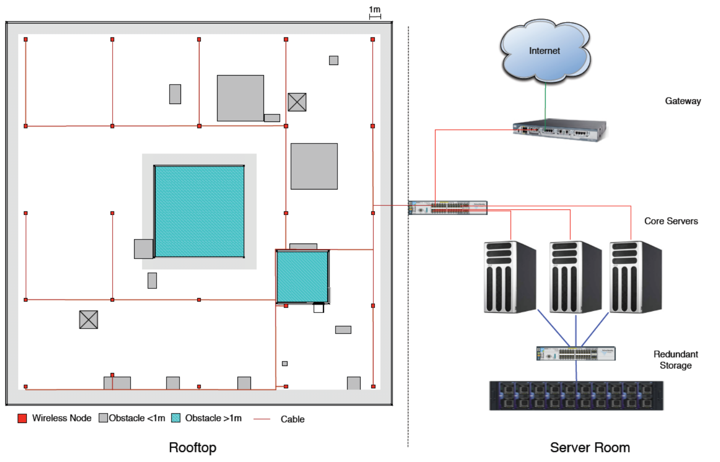


Figure 3.9: Testbed Deployment Plant

Ubuntu). New network (WiMax, 3G, LTE, etc), extensions were not planned. Nodes have their internal storage and additional redundant storage through NFS. Custom power management is made through the router, the power of which is controlled through the Ethernet Interface. Core servers support the testbed software infrastructure. More details can be seen here [70] such as board map kernel modules and tools used. Validations were made in order to assert the minimum interference achieved by the testbed. Figure 3.9 illustrates the current testbed hardware architecture.

Table 3.1: Node hardware specifications

CPU	VIA EDEN 1Ghz
RAM	1GB
Wireless Card 1	Atheros based card 802.11 a/b/g
Wireless Card 2	Atheros based card 802.11 a/b/g/n
Ethernet Card	Realtek Gigabit
Local Storage	4GB Flash Device
Remote Storage	4TB Redundant NFS Storage

AMazING testbed uses OMF as its management software, with some modifications and extensions. The CM was extended to AMazING custom power management solution. The measurements module of OMF was also extended to support “generic” metrics provided by IPFIX[71] while making usage of OML measurement infrastructure and OEDL capabilities to instrument applications. The OMF deployment of AMazING differs a bit from usual, as illustrated in Figure 3.10. Both AM and EC are located on the same machine, and the router is linked to the switch that interconnects all twenty four nodes.

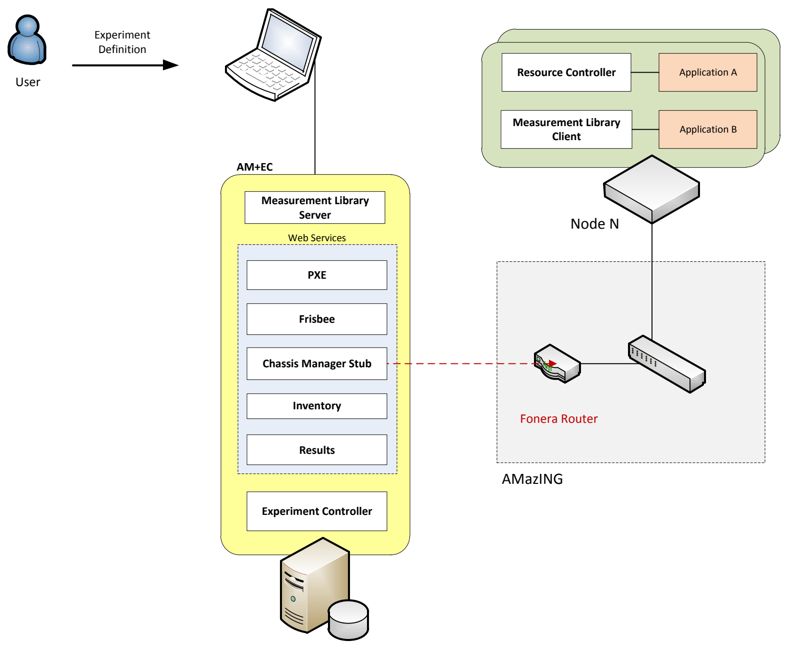


Figure 3.10: AMazING - OMF Software Deployment

3.2.2 Chassis Manager

AMazING custom power solution was implemented through a router, which was also modified in order to accomplish this capability[72]. The router exports a web service that will vary the voltage transmitted in their network interfaces. Nodes power will be then switched off independently of their underlying operating system. This is a low-level process that tries to replicate the functionality of the CM hardware, supported OMF. Both router and nodes hardware were modified to accomplish this power management capability.

As explained previously, the AM provides a stub for devices without CM capabilities. This stub comes without implementation so that testbed owners can extend it to their own custom power management solutions. The implemented stub in this testbed will contact the web service, to switch nodes power. A web application was created, called Relay Panel (Figure 3.11), to monitor testbed nodes power. It displays a testbed grid, enabling node power through the click of the nodes, making use of the router web services to control nodes power.

3.2.3 OML IPFIX Extension

Carrying out experiments on a testbed implies that the user gathers metrics, either from application itself, performance specific or protocol-related. The main problem when gathering these metrics is that usually involves using different applications, because of the different

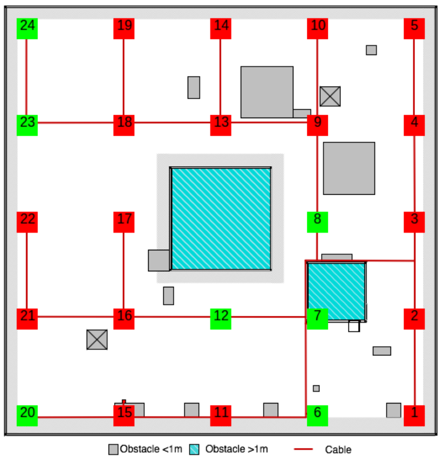


Figure 3.11: Power Control Web Application - Relay Panel

metrics they can provide. Furthermore, post-processing of the gathered metrics is often needed for later analysis and visualization. This introduces a lot of overhead in network testing and can be a slow process. OML solves most of these problems, specifically the log fragmentation and results collection. The user only needs to integrate their application with OML, create the application definition and run the experiment. For user developed applications, the integration is easy enough, but for third-party applications with a large codebase, it becomes a slow process.

To solve this problem, AMazING OML was extended to provide "generic" metrics for experimenters. In this extension, a large set of metrics is provided either metrics related to performance, or protocol-related. Most of these metrics can also be found in the previously referred applications, thus achieving a generic solution that can be used in any test scenario. This work was performed as described in [73].

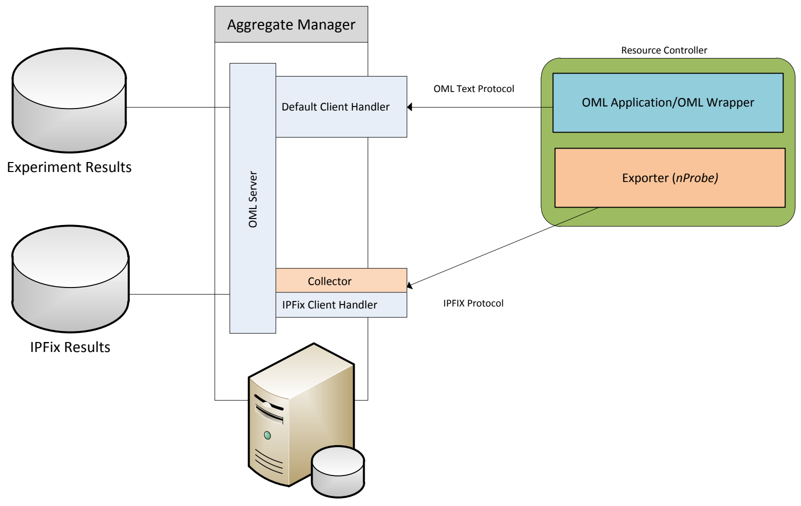


Figure 3.12: AMazING - OML architecture with IPFIX support

To support this functionality, an approach based on the IETF IPFIX standard was followed. It requires the addition of two additional entities to OML architecture, to support this IPFIX method. An Exporter will be reporting information of packet flows to a Collector. IPFIX will be the export format describing these packet flows. The Exporter is launched on the experiment nodes, and the Collector, located at the OML Collection Server, will be handling these flows information and merging them to the experiment results database, which will be provided to the user, like any other OMF experiment. Being IPFIX only an export format, metrics provision depends on the implementation of the Exporter. *nProbe*[74] was chosen, having a large set of metrics available to experimenters. The *nProbe* is integrated like any application to be used with OMF (i.e. the application definition), with one of its parameters being the template containing all the metrics to be collected. The *nProbe* will report to an IPFIX Collector (which is part of the OML Collection Server) and the results injected into a separate database. Beyond the flow information, the final database will also contain the exporter's information, the template and experiment information. The implementation was made through the extension of the server Client Handler that receives measurements from nodes.

Supported

The AMazING deployment is always updated with the more recent versions of OMF/OML, which affects this extension code. The main drawbacks of the IPFIX extension were inside the OML server. The generated results were always injected to the same database, being unable to associate to the correct experiment ID. Furthermore, a Thread remains always open even if IPFIX method is not used. Currently, this extension is being validated and asserted to be included in the main branch of the OML and supported in more recent versions of OMF/OML, installed in the AMazING testbed. Due to the previously mentioned architecture problems and minor compatibility issues, this OML extension was not given support in the proposed solution explained in the next section.

Chapter 4

Proposed Extensions

The AMazING testbed is deployed with OMF providing generic measurement collection and a unified manner to create experiments through the use of OEDL language. It also extends the whole platform to support custom power management, and integration with IPFIX, which provides a high variety of network metrics.

Still, testers must have a low-level access and a familiarization with the control framework in order to get their experiments done. To early adopters, this may be considered a bottleneck and they mistakenly carry out their experiments manually with the illusion of this being easier than to learn the testbed tools. For users, OMF just provides a command line interface for the experiment controller with a set of commands to control the testbed. This can be a costly process and must be easier for users to focus more on the scenarios they want to reproduce, instead of the underlying tools. Furthermore, this is an iterative process until an experiment reaches a complete reproduction of the desired scenario. There is no infrastructure to manage this; additionally, when multiple users are present, management is necessary and collaboration needed. There is a need for simpler means in experiment creation, which is currently non-existent.

In this section an extension to the AMazING testbed is presented. AMazING Panel is a portal which features an intuitive way to program and schedule user experiments, while allowing multiple users to use the testbed. The system requirements are here described, as well as the architecture of this implementation, and the modules supporting its infrastructure.

4.1 System Requirements and Use Cases

This extension was created to serve future users and to enhance the functionality of OMF. It urges the need to declare a more intuitive way for them to use the whole testbed, without having knowledge of the backbone that provides the whole testbed instrumentation. The identified requirements were:

- Easy to use: The purpose of this extension is to serve the users, so creation and management processes must simple and intuitive;
- Extensibility: The portal will be in constant interaction with the OMF framework, although its foundations must have loose coupling in mind, in order to allow deployment

across different machines;

- OMF Integration: As explained previously, OMF provides different services to both experimenters and testbed operators. These services include power management, inventory information and EC resources (application and experiment definitions). All of them must be integrated with this portal, mainly for maintainability, but also to enhance the user experience when creating experiments;

- Compatibility: OMF is a project with a proper release scheme and used in many institutions to manage their testbeds. Each new version brings new features, bug fixes and more stability. The portal must be compatible with the most recent versions of this framework;

- Experiment Controllability: Users must be provided with controllability of their experiments, being able to create and reproduce a variety of scenarios. When created, the user must control all his related assets in an iterative way;

- Easy experiment creation: Although OMF infrastructure provides simpler means to create experiments it still has technical issues and some overhead, becoming difficult for early adopters of AMazING testbed;

- Testbed usage management: Multiple users are expected to use the testbed, so a scheduling mechanism must manage experiments execution;

- Isolation: Experiments must maintain a high reproducibility ratio, so external interferences can be avoided on its execution;

- Different means to modify system images: Making use of OMF capabilities to manage a testbed implies creating and modifying system images. This is usually a slow process, and faster and more optimized ways need to be analyzed in order to overcome this problem;

- Same OMF foundations: The whole framework is developed using the Ruby language and related tools. This extension will also be implemented using the same tools aiming to contribute to network community, for future extensibility by third parties;

- Publicly available: The testbed must be available to anyone, although with some controllability. Users must present their intention when using the testbed, and moderators must exist in order to guard testbed usage;

- Structured Web Application: It is important that the web application is created in a structured way, clearly dividing business logic, application behavior, and interface design/layout. Aswell as it must follow web standards' guidelines to maintain the high compatibility among browsers while being on the edge of web technologies (e.g. HTML5[75]);

- Web Standards: HTML 5 has been the most notable subject in the web community in the last few years. This extension will take advantage of all these new HTML5 features although it will not compromise application compatibility with pure state-of-the-art features in HTML5;

4.2 AMazING Architecture Extensions

Described in this section are the main concepts and relations behind portal's implementation (Figure 4.1). They are important to understand the proposed solution, future modules' usefulness, and decisions made.

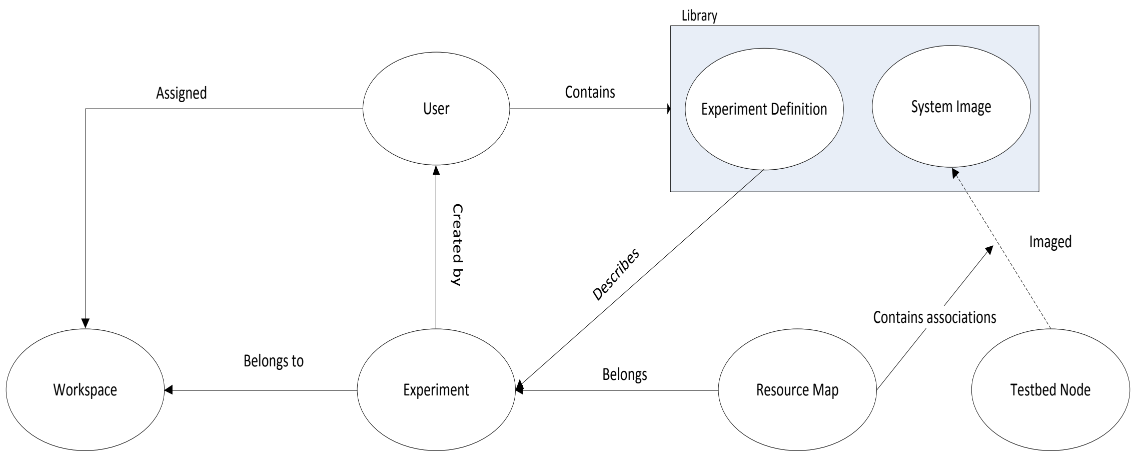


Figure 4.1: Domain Diagram

A user is assigned to workspaces, which can be either public or private. A user is an owner of experiment definitions and system images. These two resources, aggregated, comprise what is called the library. The library is an abstract container to describe all user resources. A workspace consists of a virtual space where users share their experiments and results. Normally it is created in the scope of a project or a study. Each workspace contains experiments, which are managed by its assigned members. An experiment is defined by an experiment definition and system images that describes the used testbed resources. System Images defines the disks to be imaged to nodes, to recreate their software environment. Each experiment definition depends on deployed system image software, with repositories supplying experimenters with applications for use. Testbeds provide nodes for users to execute their experiments. Resource Maps consist on the association between these nodes and system images, describe the software deployment of an experiment.

The extension made to the architecture (Figure 4.2) is the creation of the portal, a web application for users to instrument the testbed. It provides access to a library, and easy management of their experiments. This is done in a transparent way, decoupled from the rest of the OMF components, namely the AM and EC. Users now provide experiment definitions and system images that will be maintained and reused in this new portal. Both the portal and AM share the same database, being able to manage AM inventory. The portal provides a permission system to control the portal access in the testbed. Experiments are executed through a scheduling mechanism in the form of a queue. It maintains the appropriate data on users, experiments and testbeds in a repository, which is called the portal inventory. Experiments are under version control, to increase their flexibility when defining their scenarios. Furthermore, this portal is intended to abstract the users from OMF tools, while giving the same

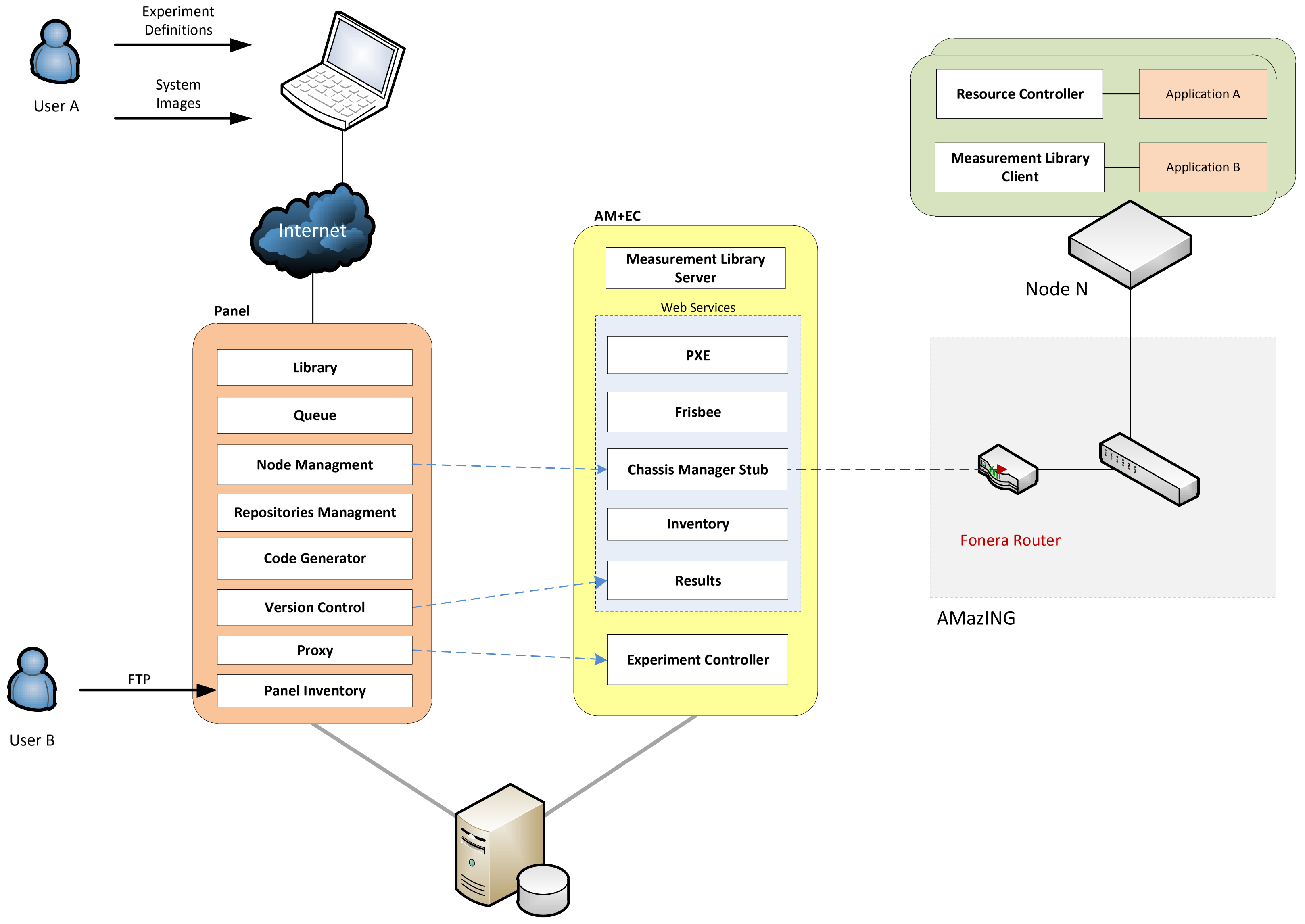


Figure 4.2: AMazING architecture extensions

control when using low-level tools. System Images contains the proper software to reproduce a desired experiment scenario. External access (via Experiment Version Control (EVC)) is also provided to users from them to change their system images.

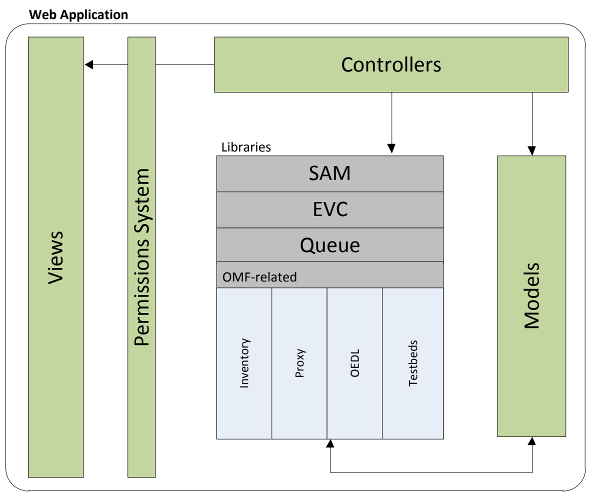


Figure 4.3: Web application internals

The web application was developed in Ruby using the Rails[76] Model-View-Controller framework: web pages are rendered through Views; Models provide the proper abstraction between the application and the database[77]; Controllers manages client responses, receiving user input, to instruct models and present the views. Since OMF is implemented using Ruby and related technologies, Rails was the best option fitting the identified system requirements. This approach enables testbed owners to easily extend this extension for additional functionality. Moreover, taking advantage of the Ruby ecosystem, certain parts became easier to implement such as the Queue and OEDL environment modules. Additional Libraries were created to complement the lack of APIs in the OMF components: inventory management and testbeds CM control. The System Accounts Manager (SAM) module was created to manage EVC users, and EVC to manage experiment versions. OEDL module powers the backend for scripts analysis/auditing with an environment that will support the Integrated Experiments Environment (IEE), to help users to create experiments. A proxy represents the mediator entity between EC and the web application. A queue manages experiments globally, while this Proxy manages them individually. In the next sections each of these modules will be explained. The web application will be described by its key features and modules supporting it.

4.3 Data Model

Since EC and AM are sharing the same resources, the low footprint on the database was taken into consideration. AM maintains the inventory of the testbed, while the portal only stores information about experiments, users accounts and library. Figure 4.4 illustrates the data model supporting the application.

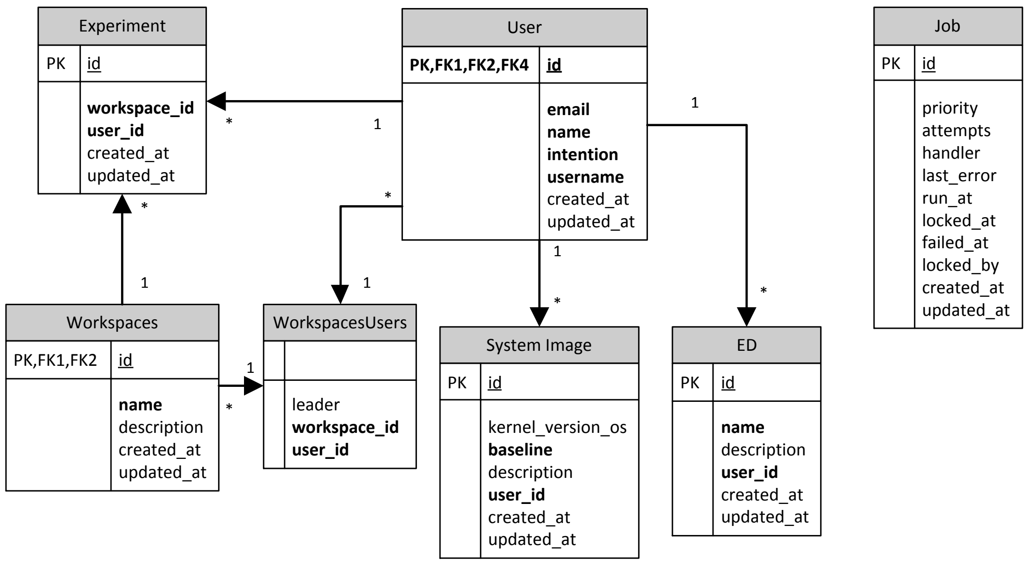


Figure 4.4: AMazING Panel Data Model

Users are associated with almost every resource. The library is an abstract concept to describe the collection of all user resources. For now, the resources are the system images and experiment definitions. An experiment is also denoted by one creator, and assigned member of a workspace. Experiment definitions are the OEDL scripts, and system images those to load into nodes. System Images have additional attributes such as Kernel Version or Baseline, although dependent on user input. Baseline is an association that indicates from which system image it was created. Users are able to download system images, extending them to their own needs.

Workspaces are virtual spaces where experiments are aggregated. It can have various users working on a workspace and users can have many workspaces, so a many-to-many relationship was needed here. Users are also distinguished as leaders or ordinary members of a workspace. Leaders will manage all experiments in the workspace and are able to assign or unassign new users to workspaces.

Jobs table are related to background jobs that might persist on the web application. These so-called jobs support long-running tasks such as email sending, image dimensioning, etc. Its implementation was made to support any task however it is only serving experiments execution. This approach is explained in subsection 4.9.2.

4.4 Permissions System

A permission system was created to control overall actions, although due to the purpose and number of actions available in the web application, a dynamic permission system was not necessary. Permissions are defined programmatically and are divided in different roles that are assigned by an admnistrator. Each user has an attribute called roles mask, representing the roles assigned. Each bit of this attribute corresponds to a role. The number of bits of this mask will be the number of the existent roles. It is possible to assign more than one role to a user, and the ones available are:

- **Experimenter**: When users are activated, this is the role assigned by default. One experimenter can manage their workspaces, library, experiments, check testbed status and see which experiments are in queue. Optionally experimenters have a sub role, ”Workspace Manager”, allowing them to assign/unassign new members to a workspace and manage all their experiments. This sub role is assigned by default when the user creates a workspace, or another manager can later assign it manually.
- **Testbed Maintainer**: This role is more of an internal one, for those with physical access to testbed nodes. Their responsibility is to notify the nodes that are under maintenance, for experimenters to know which nodes are safe to use.
- **Administrator**: He has access to everything, although navigation is carried out as if it was an experimenter. They are able to manage user accounts, permissions, and activations.

4.5 Repositories Management and Auditing

The AM, EC and Portal share the same hardware resources, meaning that OMF system images and scripts are in the same file-system. In an OMF testbed, when a user executes the load command of the experiment controller, the image filename must exist on the AM system images directory. The AM will only upload images from that location, being the testbed owners able to retain control of the software deployed on nodes.

The EC also has an additional repository, dedicated for experiments. This repository will store prototypes, application definitions and OEDL “libraries”. These “libraries” are system OEDL scripts, normally executed before the user script (e.g. image node, event initialization).

For better resource management, the portal will manage both of these repositories and two additional ones provided by the portal application. These are the portal inventory and experiment version control repositories. When adding a system image to the library, it will link this system image between the user library and the AM. For OEDL scripts, it will browse into EC repository for available OMF and user applications. Each user has a home in this EC repository. When the experimenter declares a new application to use, the portal will store their generated application definition in his repository home.

4.6 Portal Inventory

The Portal Inventory maintains all user libraries and testbed information. This inventory will maintain all user system images and experiment definitions (i.e. the user library), experiment files and testbed configuration files (see Figure 4.5). Each time an experiment is created a folder dedicated to that experiment is created in the portal inventory. The folder contains experiment log files and database results, organized by the run number of the experiment. For users, a home is provided for system images and experiment definitions storage. These system images will be linked to AM repository, hosting the system images as explained previously.

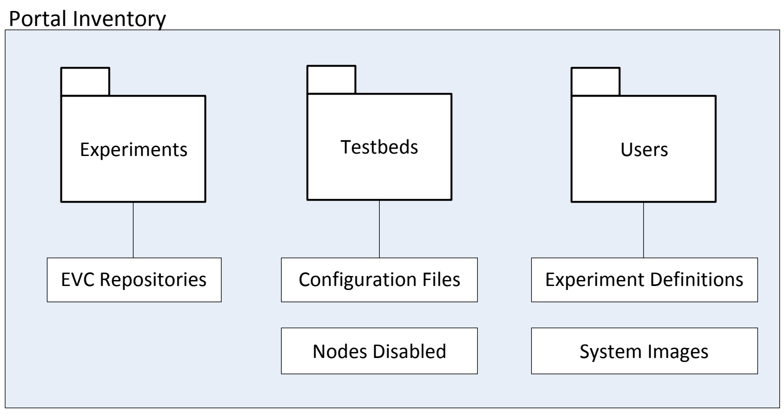


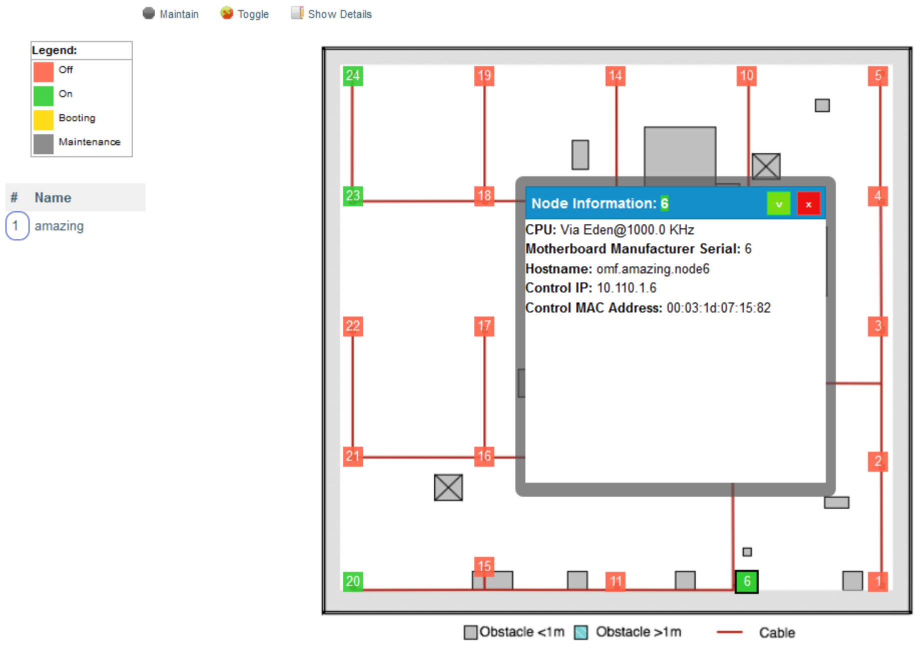
Figure 4.5: Portal Inventory Structure

Besides user resources and experiment information, this inventory stores the configuration files of AM inventory testbeds. As explained previously, the AM inventory stores information on available testbeds, testbed nodes and locations. In the portal inventory, this information is used to display nodes information. Normally a testbed can be displayed with a map with node locations. These configuration files contain nodes position based on an image, which represents the testbed map. If no testbed configuration is provided, a table of nodes is rendered instead. Apart from the testbed representation, the nodes that are disabled by the testbed maintainer will be also stored.

4.7 Testbed Management

As explained previously, AM provides a set of web services that allows an EC to control, boot and manage nodes. It was also seen that the AM provides nodes' power management either by hardware or through a custom solution which we refer as the CM stub. AMazING testbed had implemented this stub, where power is managed through a router, accessible via a web GUI called the Relay Panel, as explained in subsection 3.2.2.

Portal has integrated the Relay Panel, as shown in the Figure 4.6. It was refactored with additional features and extensions. Now it also shows hardware and network information about the node (Control IP, MAC Address, CPU, and Motherboard Information) fetched directly from the AM inventory. Colors represent different states of nodes: On, Off, No Access and Maintain. All roles can see both node state and information. Testbed Maintainers and Administrators can disable nodes, which normally happens in case of failure. As explained previously, Portal inventory powers the testbed visualization where nodes are positioned according to an image map. Extensibility is gained through this kind of approach, enabling other testbed deployments to choose their own visualization.



Node Information: 6

CPU: Via Eden@1000.0 KHz
Motherboard Manufacturer Serial: 6
Hostname: omf.amazing.node6
Control IP: 10.110.1.6
Control MAC Address: 00:03:1d:07:15:82

Figure 4.6: Refactored Relay Panel with maintain action to disable action and also node infromation displayed

4.8 Queue

OMF is only prepared to manage one testbed. The AM will provide resources, where the user is allowed to use. Concurrent experiments are supported by OMF, as long as the used resources are disjointed. When multiple users are using the testbed, there is no way to manage its use. Supporting the Portal a First In First Out (FIFO) queue was developed, to allow one experiment to be run at a time. Despite beforehand verifications that can be made to allow concurrent experiments, this approach is preferred to avoid any interference that could introduce error in experiment results.

A job scheduler was used to enable this FIFO queue behavior, following the Worker Design Pattern. The scheduler will spawn a preset number of workers which periodically polls the database for jobs to run. The Job table illustrated in Figure 4.7 maintains the information that workers will need to execute experiments. Once the user queues an experiment, a row will be added to this table. Here is described when to run the experiment, and more importantly the handler class that the worker will dispatch the job. This handler class is responsible for the experiment execution. There the experiment will be prepared and executed using a Proxy to communicate with the OMF testbed. The Proxy will be explained in subsection 4.9.2.

4.9 Modules

A set of developed modules supports Portal functionality. Aside from the Business Logic, and Client implementation, these modules are the most important parts of the Portal, designed to fulfill the identified system requirements. These modules include:

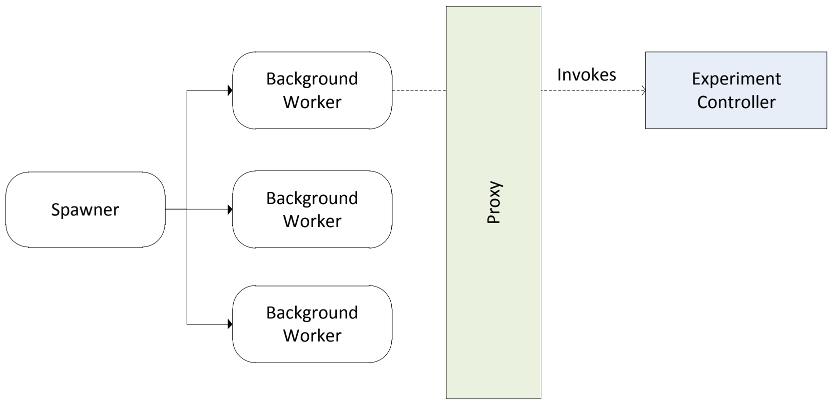


Figure 4.7: Interaction Queue components with testbed

- Account management for integration between Portal and File Transfer Protocol (FTP) accounts;
- Versioning control of experiments, integrated with the Portal;
- The proxy module, which is in charge of the interaction with EC.
- An Integrated Experiments Environment, similar to IDE functionality but for the experiments definition creation.

4.9.1 System Accounts Manager

System Images and Experiment Definitions are the key parts of the experiments creation. After successive runs of the same experiment, different versions of the application may emerge due to bug fixes or operating system deployment. This implies that the system image must be constantly updated in order to support the desired scenarios for an experiment.

Besides the HTTP upload available to update the system image, access to user library is also provided through more adequate methods. FTP access is provided for users to change their system images, without having a system account in the host machine. Instead of having a system account on the host machine, the FTP server will manage access through virtual users (Figure 4.8). This way users will get only access to the FTP service.

The chosen FTP server was pureFTPd[78]. Many features come with this server such as MySQL authentication or the previously explained virtual users. These are managed as normal UNIX accounts, although with additional options such as quota management, bandwidth available and download/upload ratios. Virtual users' management is made through UNIX like system accounts, although limited to FTP with special features. Example 2 shows which options are available for the configuration of these accounts.

FTP access is directly controlled from portal account users. The SAM module is the bridge between Portal and FTP service. When the user registers on the portal, an inactive FTP account is created. Once an administrator validates the user, his FTP account will be further activated.

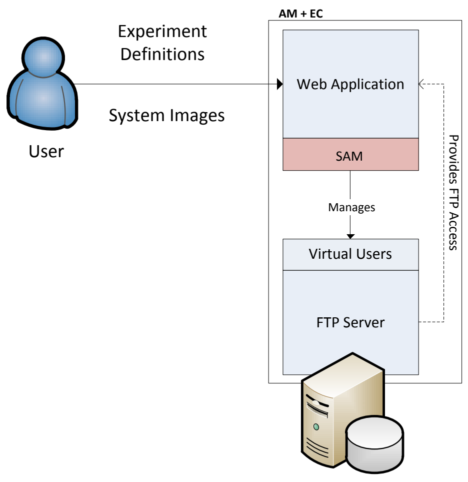


Figure 4.8: PureFTPd and SAM workflow

Example 2 pureFTPd User Accounts

```
<username>:<password>:<uid>:<gid>:<gecos>:<home directory>:<upload
    bandwidth>:<download bandwidth>:<upload ratio>:<download ratio
    >:<max number of connections>:<files quota>:<size quota>:<
    authorized local IPs>:<refused local IPs>:<authorized client IPs
    >:<refused client IPs>:<time restrictions>
```

4.9.2 Proxy

One of the concerns related to the Portal development is to have all components explicitly decoupled, to promote the modularity of the web application. Modules involve different parts of the web application, including accounting just explained in the previous section. More importantly, OMF interaction must be well isolated, for the Portal implementation to be decoupled from the testbed operations. The Proxy role here is to mediate the interaction between the application and the testbed.

Proxy abstracts the web application from the testbed in two manners: experiment execution and preparation. Both of them describe a behavior that the testbed will "expose" in the experiments execution. OMF does not provide an API for developers to programmatically use the testbed, instead users have the EC CLI. Users without the portal must issue a "load" command in the terminal for each image they want to load into nodes, followed by an "exec" to execute the experiment. The proxy role here is to make this process transparent to the web application. The assets (system images, nodes, experiment definition) will be passed down to this proxy, being its responsibility to communicate with the AM/EC/RC or by other means if needed. The Proxy API exports "prepare", "start", and "run" routines for the web application to use. Behind these routines, state machines describe the preparation, and execution(s) behavior of the experiments. These state machines consist of a sequence of actions that must be implemented by the inherited proxy. An AbstractProxy bundles the implementation of the behaviors, but with an empty implementation of his contained actions. The ones that extend this AbstractProxy must tackle on these actions for experiments to be executed. Since Portal, AM and EC reside on the same machine, a LocalProxy is provided, where the implementation will execute the CLI commands.

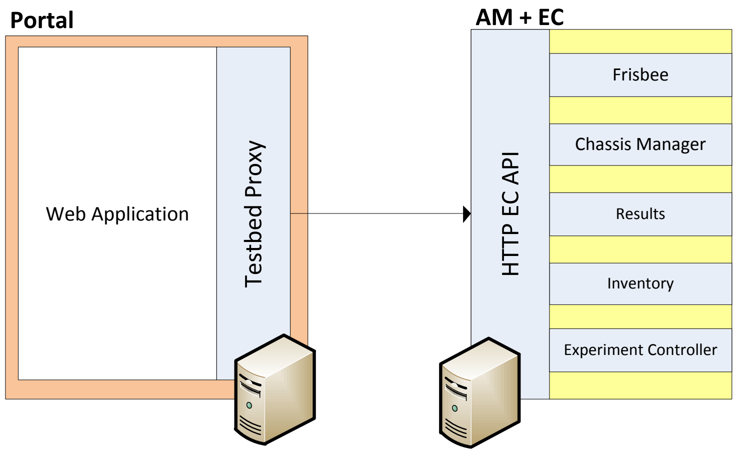


Figure 4.9: Proxy Usage Example

This approach has many advantages for the extensibility and modularity of the web application. OMF Interaction is clearly divided from the business logic of the web application. The Proxy will take care of the whole testbed interaction, while the testbed owner is free to use any means to communicate with the testbed. For instance, one could develop an HTTP Proxy that could communicate with a multiple remote OMF-enabled testbeds, as illustrated in Figure 4.9). The alteration of this proxy would make no changes in the Web Application logic.

The behaviors the base Proxy exposes are for preparation, initiation, and single and multiple runs (hereby referred to as batch run). While an experiment is being executed, it also maintains an overall state, which continuously advances from its preparation to its conclusion. All of them change the experiment state, according to the invocation of the actions. Run is a composition of the preparation and initiation. The batch run consists on multiple invocations of the run routine. These behaviors are state machines (Figure 4.10 and 4.11), with the following sequence of actions:

1. Generates OMF experiment ID, using number of runs and portal experiment ID
2. Remove unneeded files (e.g. logs).
3. The experiment starts preparing, resources loading takes place. A resource map is needed for this action to take place. As explained previously, the resource map contains associations between system images and nodes. This action will load each system image to a set of nodes.
4. This action is implemented by the AbstractProxy, referring to internal states of experiment based on the OMF log files. If successful, the experiment is now prepared.

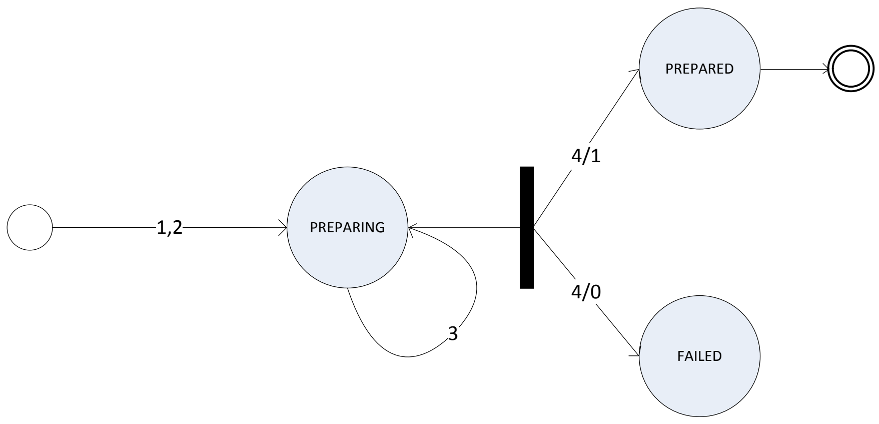


Figure 4.10: Proxy Preparation State Machine

5. Generates OMF experiment ID, using number of runs and portal experiment ID.
6. Generates the author's file. It contains the run number, experiment version and author.
7. Experiment starts. The OEDL script is chosen by its version and using the previous generated ID.
8. Results are fetched from testbed.
9. Run is saved with results and experiment log files.
10. Author's file is removed, since experiment finished.
11. Internal OMF state checked and if successful, results are copied to the portal inventory folder. Upon its completion, the same experiment remains prepared to be executed afterwards, if the user desires.

Single and batch run are a merge of preparation and execution behaviors. Preparation is done only if needed and successive runs are made afterwards.

4.9.3 Experiment Version Control

One of the benefits of an OMF-enabled testbed is the repeatability that a user gets when describing his experiment workflow. With the Portal, an experiment can easily be re-run, bringing the automation of creating it once and executing as many times as the user wants.

The user faces some overhead in their experiment creation. Some of the issues include software management, multiple experiment scenarios and eventually node physical failures. Users of the platform may also experience difficulties when creating scripts, due to the OEDL learning curve or successive scenario changes. The experiment is mostly defined through a script and a set of disk images to be deployed on nodes. As explained previously, the OEDL script holds the description of resources, network configurations, and application initiation.

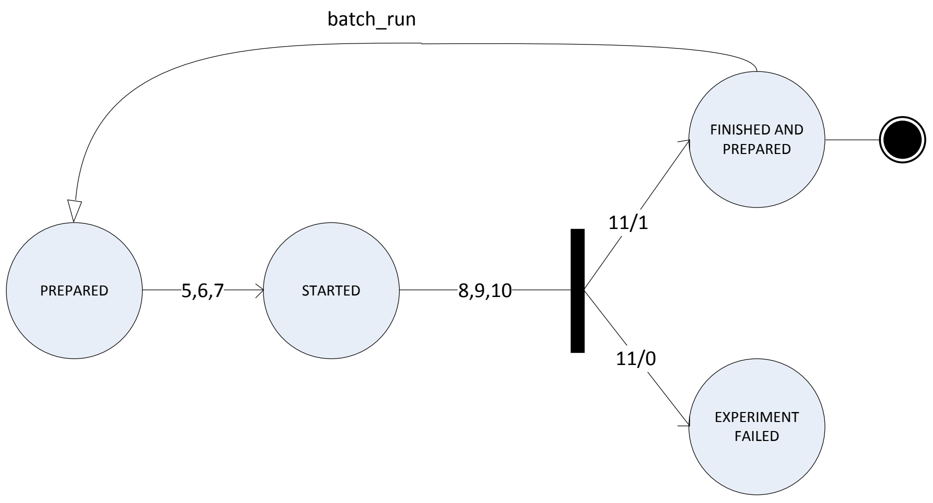


Figure 4.11: Proxy Execution State Machine

Since the script is made through a programming language, its correct execution depends on both syntactical and semantic validation. In addition, the experiment script creation can take various iterations of a desired scenario until it gets to the final version. Furthermore, the change of the resources map, also inflicts script changes. The Workspace is used to organize experiments within a workgroup, as explained before. The context of these workspaces is often related to a project, a scientific paper, application test, etc. For different contexts, users may separate different scenarios in one experiment, by simply changing application parameters, choosing different resources to use, among other possibilities.

All these reasons urge the need of a Version Control System (VCS) to support experiments. It greatly increases the productivity of the user, providing a much better management of their experiments. The whole environment supporting experiments enables capabilities such as the fallback to a more correct scenario of an experiment; easily switch the scenario for better results; much better organization instead of flooding workspaces with "same purpose" experiments. There are several options in the versioning control systems. All of them pose different approaches, some centralized[79] and others distributed[80], although they share a high-level complexity in terms of versioning control. Such larger infrastructure is not suited for Portal requirements.

A custom VCS, called the EVC, was developed. Its main requirements include: basic versioning control (branches, commits), manage results by keeping track of them per run and easy access to all repository information either metadata or the assets (scripts, logs, results). The Architecture follows the object model illustrated in Figure 4.12. 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. Additionally, a folder contains a set of runs, each one containing log files and database results. Branch information includes commit logs and the number of successful/failed runs. All this information is stored using YAML files. Security was not a concern since access to the testbed host is not provided externally. Storage optimization was not a priority so each branch will maintain a snapshot

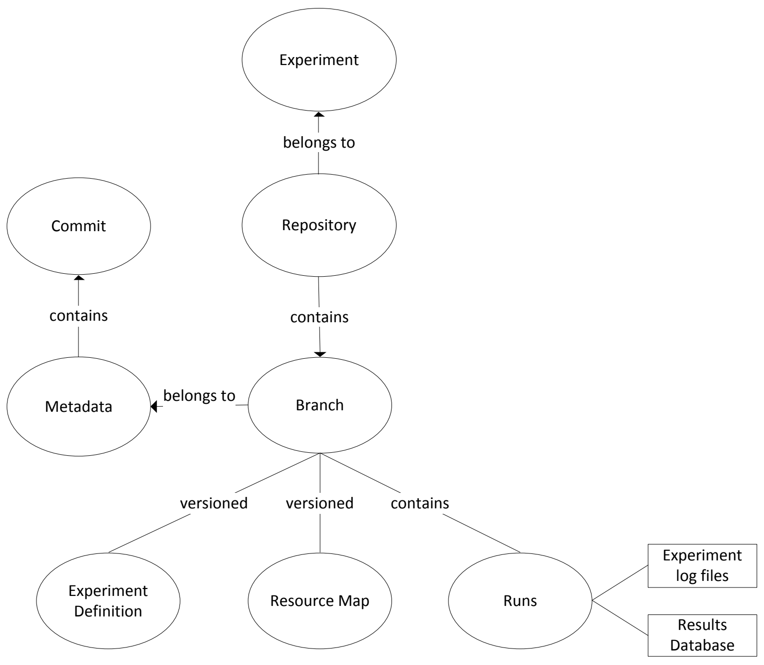


Figure 4.12: EVC Object Model

of all the assets when a new revision is made.

4.9.4 Integrated Experiments Environment

As explained previously, in scripts lie all the experiment's workflow and logic. Here are located application initiation, resource configuration, nodes to use, topologies to use, applications deployed in nodes, etc. General programming skills are required, but with less effort in learning Ruby itself.

The main advantage of using OMF is the automation gained when creating and executing experiments. Users do not need to manually deploy and run software being able to automate to whole process, from the configuration of the experiment until the obtained results. Although creating experiments requires much knowledge of the tools underneath the testbed. The user must understand the OMF entities role (AM, RC, EC), as well as the orchestration process it involves before creating the experiment (creating/loading images, script creation, nodes initialization). Although OEDL was made to facilitate users, the learning curve to create the script can slow the whole process, not encouraging new users to adopt the platform and use it. Usually when it comes to big projects or big software packages, establishing scenarios and creating experiments may need a more hierarchical, organized infrastructure, therefore needing collaboration tools, experiment description enforcement, and ease of user adoption.

Integrated Development Environments (IDEs) are well known among the computer science community for being an important tool that simplifies these tasks, and also motivate the developer to be organized. To solve the problems stated previously, a similar approach to these IDEs was followed. Users can create their experiment definition, through a GUI called

IEE. The user can always see or edit the script that is generated, to help OEDL learning. Furthermore, by having this kind of environment the user, can better understand what capabilities are available for an experiment, instead of having to read the OEDL reference. Script creation difficulty decreases, suggesting users to think more about the scenarios they want to represent, in their experiment. Its purpose is only to help users in the creation of their experiment definitions.

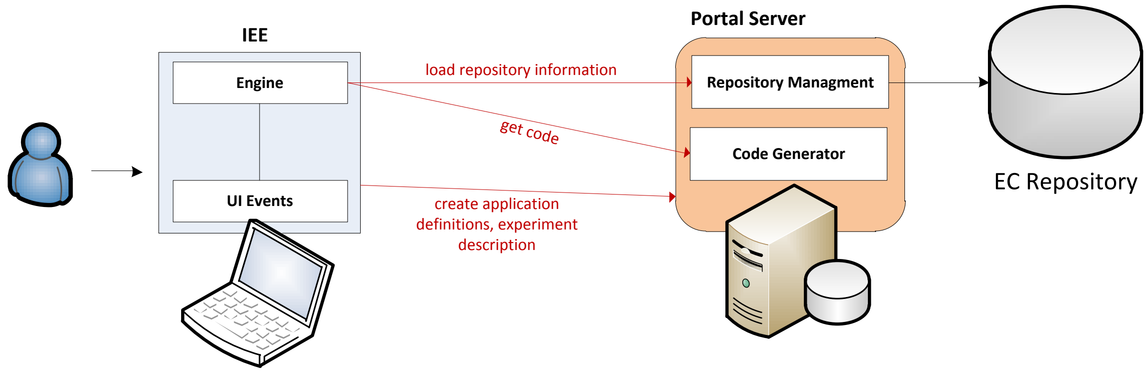
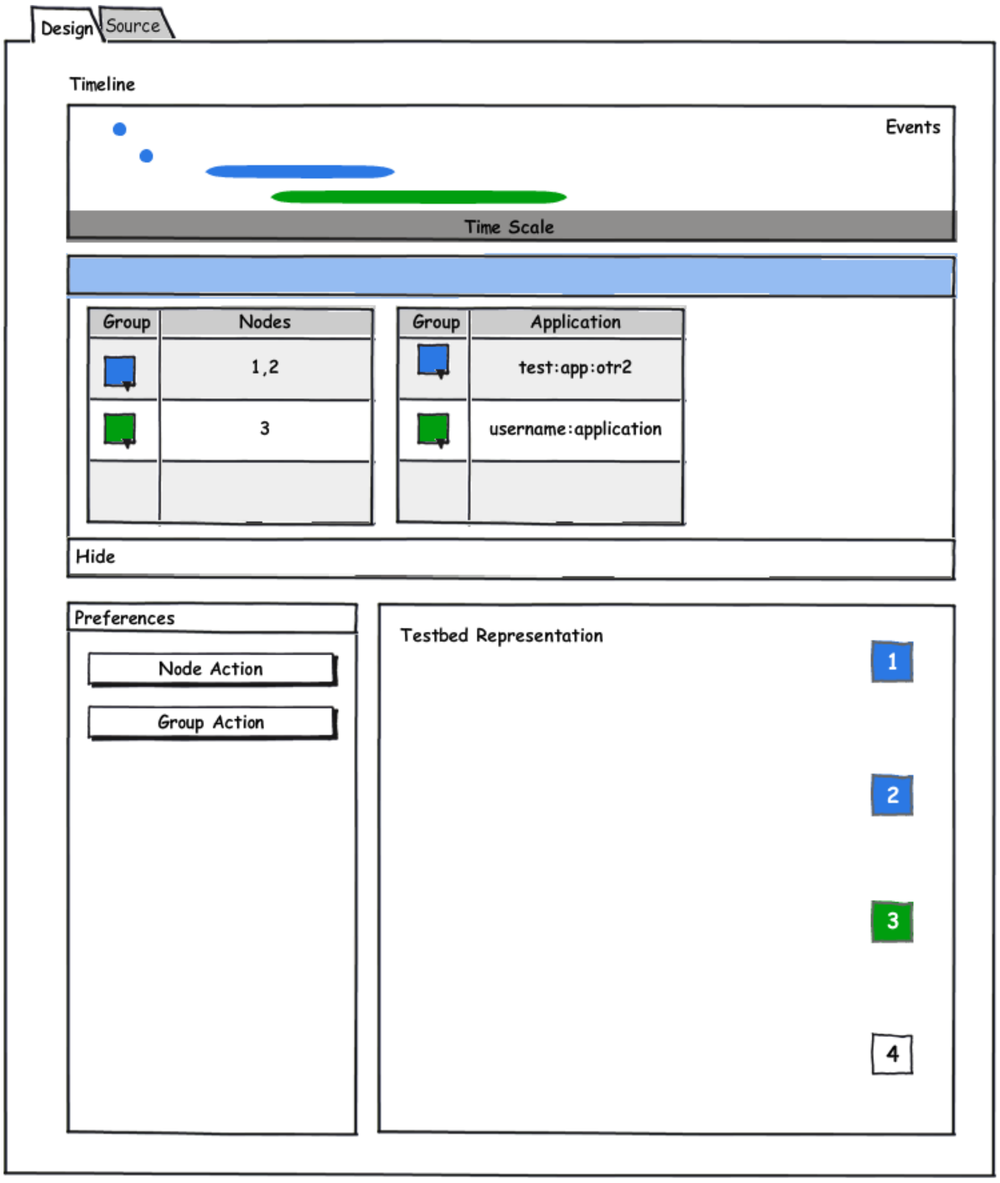


Figure 4.14: IEE Components

The IEE GUI is based on the testbed visualization (Figure 3.11 and 4.6). The proposed layout is displayed in Figure 4.13. On the left sidebar, available options are presented, by selecting nodes with the same color. It is the same approach as the resource map definition; to better identify the nodes being managed. The tables present both applications and groups defined on the script. As mentioned previously, the user can always see the script code, by selecting "Source" tab. A timeline on top presents the execution time flow. There, a user can execute commands or start their applications. Clicking on a group tables, the user is able to select a timestamp and give a command as input. The same goes for applications, in which the input is the duration to execute all applications on the group. By selecting an arbitrary set of nodes, the user is able to define their network properties and applications.

IEE design principles are similar to Rich Internet Applications[81]: the client manages the entire content and user interface, using the server only for Remote Procedure Call (RPC) calls, to manage data. The main advantage of this approach is the low server usage. The architecture of this solution is divided into two parts: the presentation layer and script logic. The view is responsible for the general styling, event handling for the user input and timeline generation. Supporting the view layer script generation, repository auditing and user application management is enabled through web services, exchanging data using the JavaScript Object Notation (JSON) data format. The IEE fetches the OEDL applications reference, containing all repository information, specifically the applications, provided by OMF or the user. When a user adds an application, all information about its properties comes from this reference. It also maintains information according to user actions: defined groups, applications, network properties and application properties. JSON data is generated with all this information, using web services to fetch the experiment definition code.

The IEE View layer is divided in two different components:



Group	Nodes
🟦	1,2
🟩	3

Group	Application
🟦	test:app:otr2
🟩	username:application

Figure 4.13: Proposed IEE Layout Wireframe

- The Engine responsible for handling all data related methods. It keeps the state of the environment maintaining information about groups, timeline events, resource properties, and applications

- The UI component is responsible for the input handling sending the appropriate data that the engine will maintain during the experiment creation. This component is divided between content generators, helpers and templates used to generate the proper HTML.

Supporting the IEE web services, an environment that defines portal's OEDL implementation was created to audit scripts, scanning EC repositories and user experiment definitions. The experiment definition will be generated according to user actions in the IEE GUI. The code generator will be responsible for taking the data from the IEE Engine, and delivers the source code for the script and application definitions. In the next sections both of these supporting modules will be explained: the OEDL environment and the code generator.

4.9.5 OEDL Environment Wrapper

As explained previously, OEDL is a DSL in Ruby. It is composed by a set of routines, and properties that are able to control experiment flow and resources. In EC resides the implementation of OEDL, whereas contact between RC's is made when the appropriate methods are invoked. Following a Facade design pattern, the language is a simpler and unified interface abstracting a higher complexity, which is the management of the testbed. Its foundations lie on an environment that is bound in execution when the script is loaded. This is possible due to the introspection properties of dynamic languages such as JavaScript, Ruby or Python. A context is bound to an evaluation of a script. This context consists on a scope that is given to the script, to access local variables, classes and global methods of OEDL.

One of the main drawbacks of OMF OEDL implementation is the lack of means to access experiment definitions information. Using their implementation, information such as the groups defined, properties of the experiment, applications deployed cannot be accessed. It is important that we know which data is described in experiment definitions and application definitions to be displayed or later used in portal. Moreover, their implementation of OEDL resides on Singleton objects to store all the experiment information. Experiments executed from the EC CLI are bound to an instance, so multiple invocations in this same instance could corrupt the information contained inside those singleton objects.

Figure 4.15 shows which objects are created when the execution of experiments takes place. From the command line an OEDL script called `ed.rb` is executed. The NH, Communicator and Experiment are the singleton objects existent during execution. NH is the entry point when using OMF CLI. From this point, it will initiate the Communicator and trigger the load of the Experiment script. Communicator role is to abstract XMPP communications between EC and RCs. Experiment is the object that holds the script information. All experiments must load first two special scripts: stdlib, which will reset all nodes and wait from their response to start the experiment, and the events script to define default events that will be triggered when the user script is executed. When `ed.rb` is loaded, communication between RCs takes place, to configure network, applications and to define interaction between the applications. Information contained in the script is neither stored nor does an API exist to control these procedures. As the OEDL routines are invoked the EC will communicate to the RC'. In the portal only script information must be accessed excluding the whole

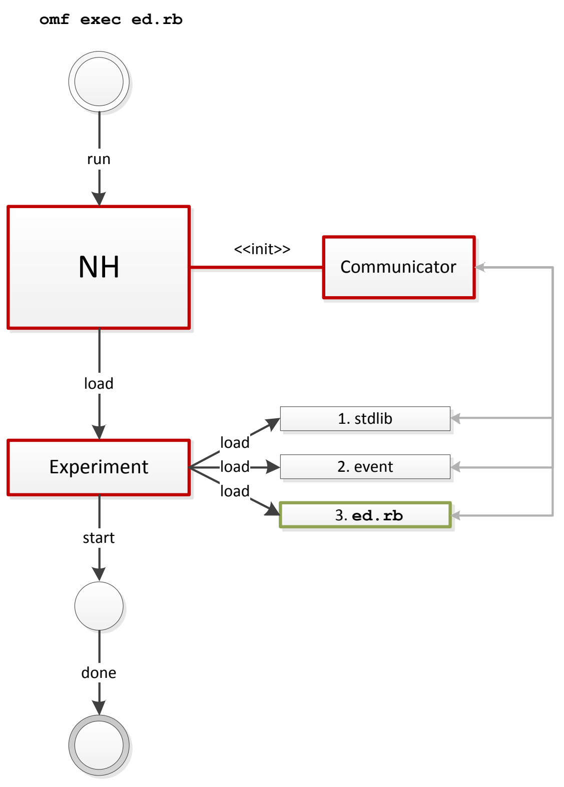


Figure 4.15: EC execution flow

communication, becoming impossible to reuse of this OEDL implementation.

Portal will be dealing with dozens of experiments definitions while managing EC repository and user experiments. For this purpose a custom OEDL implementation was required in order to fetch script information. For instance, when a defining the ip of a group (see section 3.1.3) the EC sends a configuration commands to the nodes RCs. In this implementation, no interaction between RCs is made, storing only the IP information for later use. This implementation allows the monitor of experiments while supporting the visualization of script information when using IEE (application properties, network properties). Moreover, when the user is creating an experiment, only the used nodes are displayed. Thanks to this OEDL environment, it brings this kind of introspection properties that the portal requires to access experiment definitions.

4.9.6 OEDL Generator

The OEDL language is easy to learn due to their structure and Ruby Language syntactic properties. Groups and application properties are defined in blocks, which helps users to visually understand what information is inside experiment definitions. Thanks to this structure, the experiment configuration is almost reduced to simple attributions inside these blocks. The OEDL Generator will generate code by receiving data in JSON format, as illustrated in 4.16. In the initialization of the IEE, a reference will be fetched from the server. This reference contains information about the applications definitions stored in the EC repository. Users will use this data to configure applications and properties on groups. Timeline will contain execution semantics with start and stop timestamps related to commands or applications to initiate. All this information plus experiment properties is sent to the server, where the experiment definition code will be returned and displayed in the IEE.

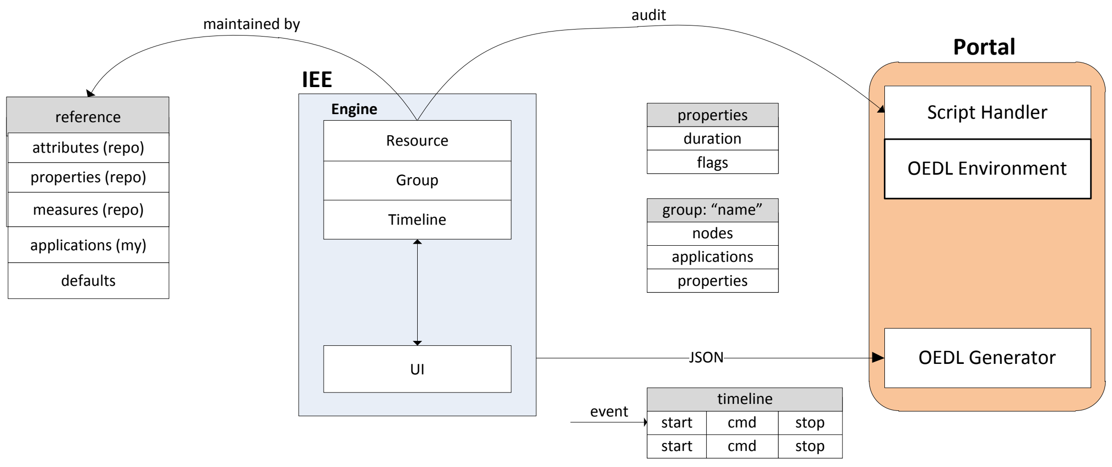


Figure 4.16: Data exchanged in IEE to generate code

Chapter 5

AMazING Panel

Until now, different approaches in terms of network testing were studied as well as existent testbed management infrastructures. Simulations provide great repeatability of experiments and controllability of environment, but with less reliability of results. Testbeds provide a way of overcoming that problem, but at the cost of testbed deployment, and its administration and configurability for users and testbed operators. Emulation provides the middle ground between simulation and testbeds. Currently, little choice of testbed management exists in the network community, where available tools are state-of-the-art. OMF from NICTA workgroup is one example, and is used in the AMazING testbed, deployed at the Instituto of Telecomunicações. However, testbed management systems are almost non-existent in experimentation community, with the exception of DES-Testbed, though it lacks important features present in the OMF framework (e.g. imaging features).

As explained in the previous section, an extension to OMF was proposed and developed, called AMaZING Panel. This extension helps users to work with OMF tools, while enhancing experiment repeatability. It consists of a web application, deployed in the AMazING testbed, destined for OMF-enabled testbeds, featuring an organized method for users to make their experiments, with version control, project management, fast deployment of their software, and an integrated environment for easier experiment creation/management.

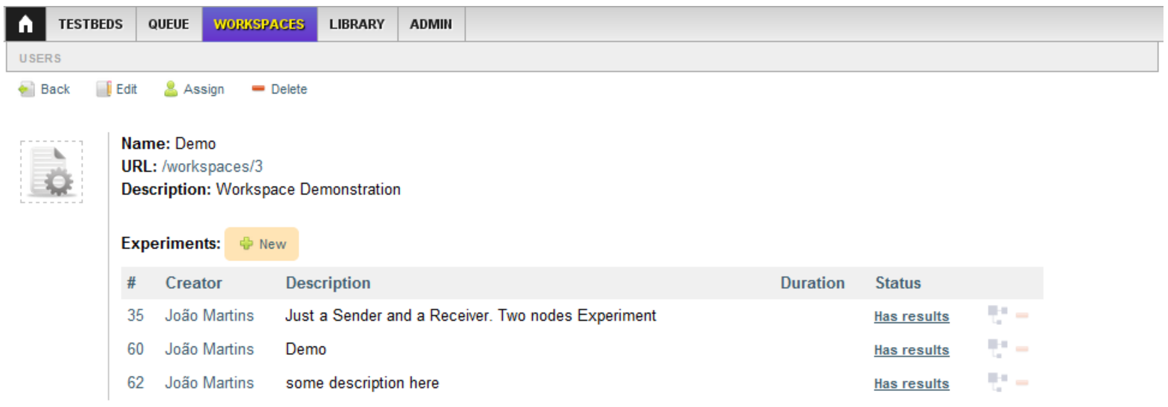
In this section, it will be shown how this implementation can greatly improve user's experiences in the experiment creation process, as well as execution and post-analysis, without requiring much knowledge of the underlying tools. The topics are:

- The complete workflow to follow when using this platform
- Different experiment scenarios
- A web application performance revision

5.1 Workflow

For a user to start using this platform, a workflow is implied in order to create experiments in AMazING Panel. The OMF workflow (see section 3.1) was extended, to one that adapts better to user needs. A typical user has an experimenter role. As it was explained, this type of user can manage its own workspaces, library and experiments. Before he creates an experiment, a user will need to upload certain assets, namely the system image(s) and experiment definition. This is all that is needed in order to create his experiment. The uploaded system image must be from the baseline that is provided, due to software dependencies persisting on each testbed node. This baseline image contains all the software components needed to interact with an OMF-enabled testbed (OML libraries, OML base applications, OMF resource controller), plus user software under test (additional kernel modules, third-party applications, software packages, etc.). Previously it was shown that an experiment definition contains all logic conducting the experiment. This includes application definitions, topologies, network properties, command line commands, etc. The Portal eases the task of experiment definition and creation, through the previous described IEE, helping in the resource and application configuration processes.

First, a user must sign up and wait for account activation (administrative process). The user must declare his intention and institution, in order to get access to the testbed. Once their account is activated, the user must create his workspace to create experiments (see Figure 5.1), either private or public. Normally, this workspace is related to the scope of the experiments, typically a project. Once created, the user automatically becomes the manager of the workspace. A manager is a sub-role of an experimenter, and is able to add or remove users from his workspace. It also has complete control of its contained experiments.



Name: Demo
URL: /workspaces/3
Description: Workspace Demonstration

Experiments:

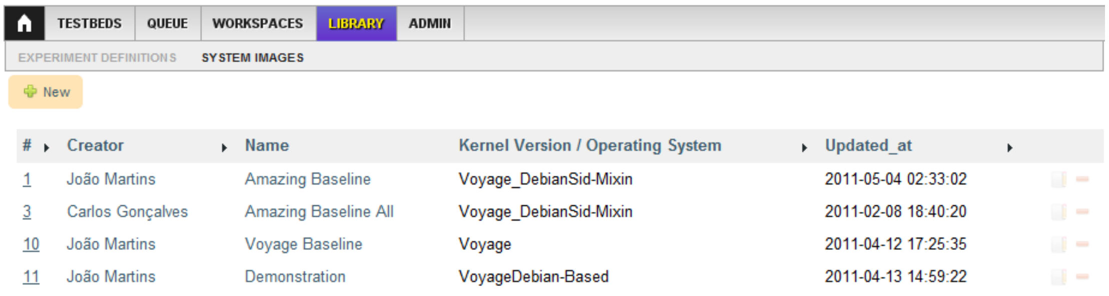
#	Creator	Description	Duration	Status
35	João Martins	Just a Sender and a Receiver. Two nodes Experiment		Has results
60	João Martins	Demo		Has results
62	João Martins	some description here		Has results

Figure 5.1: A Project Home

As explained previously, a user has a library in which all experiment definitions (Figure 5.3) and system images (Figure 5.2) are collected. All the contents are public, and can therefore be used in any experiment. Although available to anyone, only the owner of the resources can change them. An experiment definition and system image(s) are needed to create an experiment.

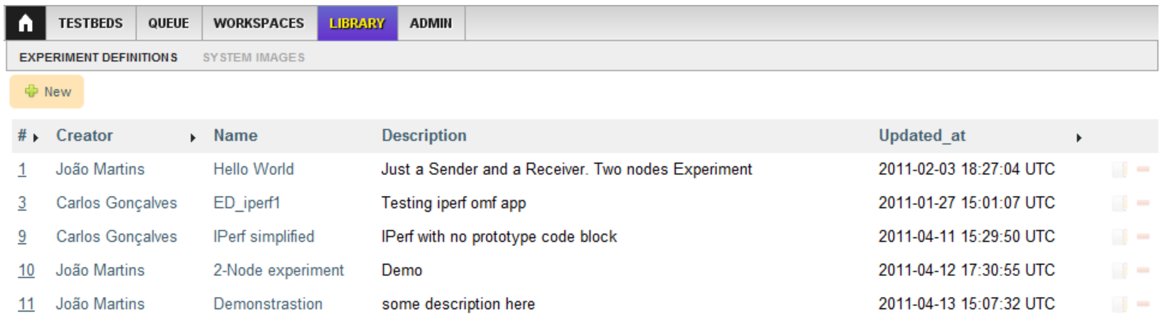
For system images, the user must supply additional information about the image, such as Operating System or Kernel Version and indicate their baseline image. For experiment definitions, an OEDL script can be provided, or the user can use the IEE. The use of IEE is strongly advised since it simplifies most of the tasks on the experiment script creation, but this will be described in detail later. Both of these library resources must provide a description that briefly explains its contents.

Once the resources are uploaded, the experiment creation takes place. On the experiment form, an experiment definition and the system images are chosen. In this step, the user will



#	Creator	Name	Kernel Version / Operating System	Updated_at
1	João Martins	Amazing Baseline	Voyage_DebianSid-Mixin	2011-05-04 02:33:02
3	Carlos Gonçalves	Amazing Baseline All	Voyage_DebianSid-Mixin	2011-02-08 18:40:20
10	João Martins	Voyage Baseline	Voyage	2011-04-12 17:25:35
11	João Martins	Demonstration	VoyageDebian-Based	2011-04-13 14:59:22

Figure 5.2: The System Images Library

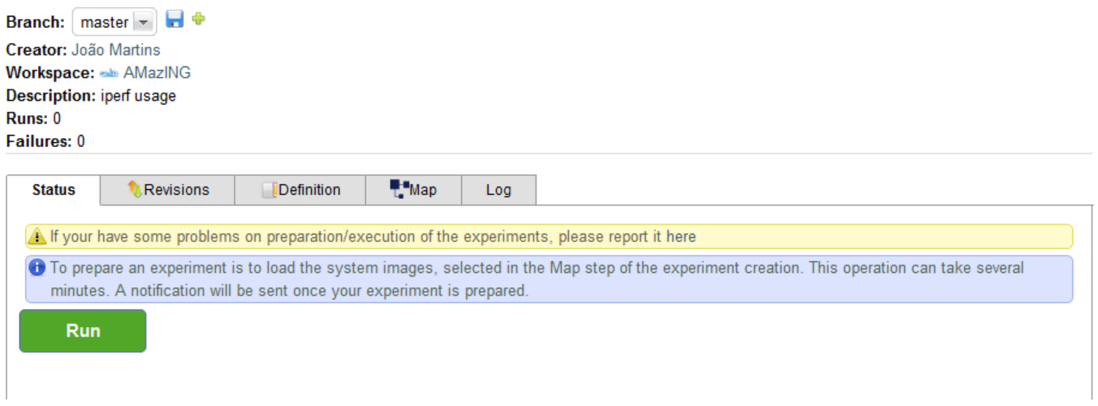


#	Creator	Name	Description	Updated_at
1	João Martins	Hello World	Just a Sender and a Receiver. Two nodes Experiment	2011-02-03 18:27:04 UTC
3	Carlos Gonçalves	ED_iperf1	Testing iperf omf app	2011-01-27 15:01:07 UTC
9	Carlos Gonçalves	IPerf simplified	IPerf with no prototype code block	2011-04-11 15:29:50 UTC
10	João Martins	2-Node experiment	Demo	2011-04-12 17:30:55 UTC
11	João Martins	Demonstrastion	some description here	2011-04-13 15:07:32 UTC

Figure 5.3: The Experiment Definitions Library

associate system images to nodes, creating what is called a resource map. The experiment definition and the resources map are separated because they are two separate processes in experiment execution.

To create an experiment it was only needed the experiment definition, system images, and the association with testbed nodes. Once we create it, we get access into an environment to manage the experiment. Editing the experiment definition and changing resource mapping at any time are some of the possible actions. The user will queue a number of runs of the experiment, and must wait until their conclusion. Once the experiment finishes the user is notified by email.


Branch: master
Creator: João Martins
Workspace: AMazING
Description: iperf usage
Runs: 0
Failures: 0

If your have some problems on preparation/execution of the experiments, please report it here

To prepare an experiment is to load the system images, selected in the Map step of the experiment creation. This operation can take several minutes. A notification will be sent once your experiment is prepared.

Figure 5.4: An Experiment managment screen

As illustrated in Figure 5.4, the main layout is organized through tabs. Each tab has different purposes. The user can observe the current status of the experiment, being able to see the load progress of images. The revisions tab allows a user to switch to another commit. Its purpose is to keep track of all commits in a branch, showing different versions of the experiment assets. The user can always change their experiment definition and resources map. Normally resources map are changed when adding more nodes to an experiment. Both experiment definition and resource map must be in conformity. The results tab is where users can create graphics with their results comparing them between different results or the user can, download the results for use in another application, such as Matlab. The last tab, displays experiment log information.

This environment motivates the user to create their experiments, and refactor them as needed, while keeping a complete track of changes and different scenarios. In the next sections different experiment scenarios will be showed. In these scenarios, more focus is given to the experiment script to show the IEE, and repeatability achieved when using version control as well as the portal capabilities to manage experiments. In the first scenario, `iperf`[82] will be used to obtain metrics related to the network performance. The last experiment shows a more elaborated experiment, describing the procedures to make when testing user-developed applications.

5.2 Experiment - Network Performance

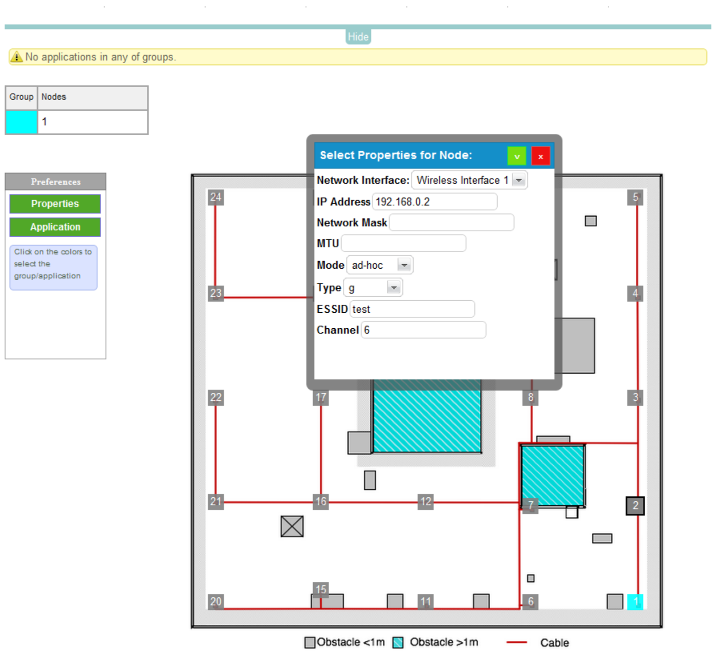
In this scenario, we will take advantage of IEE to create the experiment using one of the OML applications provided by the OMF framework. Tough, the experiment is very simple its purpose is to demonstrate basic workflow and portal capabilities. It consists of an experiment with two nodes: a sender and a receiver. The iperf, which are integrated with OML, will generate important metrics, such as Bandwidth, Jitter, Packet Loss, and Transfer Rate. These metrics are important to measure network performance.

Normally iperf is used with other modules or third-party applications, for example a routing module, where the experimenter wants to measure TCP/UDP bandwidth performance. In this scenario, UDP traffic will be generated from a client to a server. Only two nodes are used, and both in adhoc mode.

5.2.1 Resource Configuration

The experiment requires two nodes, configured in the same wireless network. OEDL organizes resources per groups, although when using IEE this is transparent for the user. You select nodes and configure its network properties, depending on the network interface.

As illustrated in Figure 5.5, network properties are set by clicking on the testbed nodes and writing the desired properties. Only those which are filled are the ones used. Once again, nodes are colored to indicate different groups. A group for the client and another for the server are automatically created by individually configuring each node. Groups are automatically created whether the user configure applications or network properties in nodes. Tables on top of the testbed map, show the existent groups and applications, displaying their color/node assignment. The left side of the testbed map shows a toolbox where it is displayed the available options for the selection of groups and nodes. Selecting nodes from different



Select Properties for Node:

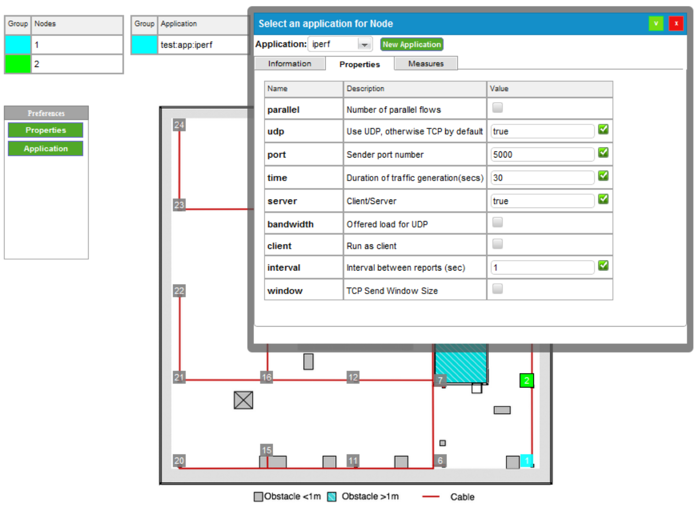
Network Interface:	Wireless Interface 1
IP Address	192.168.0.2
Network Mask	
MTU	
Mode	ad-hoc
Type	g
ESSID	test
Channel	6

Figure 5.5: Node 1 configuration

groups will not display any options in the toolbox.

5.2.2 Application Configuration

Application definitions bundled with OMF are all scanned from the repositories, and displayed to the user as illustrated in Figure 5.6. Due to the created OEDL environment the portal is able to display all this information. The descriptions of the properties are displayed for easy understanding of how to use the application. Examples are the `udp`, `port` properties configured for one of the nodes in the the experiment.

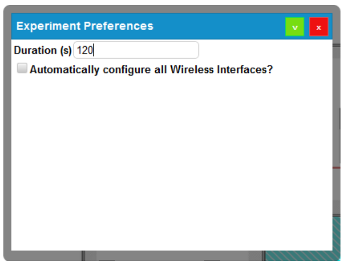


Select an application for Node

Application: iperf

Name	Description	Value
parallel	Number of parallel flows	☐
udp	Use UDP, otherwise TCP by default	true ✓
port	Sender port number	5000 ✓
time	Duration of traffic generation(secs)	30 ✓
server	Client/Server	true ✓
bandwidth	Offered load for UDP	☐
client	Run as client	☐
interval	Interval between reports (sec)	1 ✓
window	TCP Send Window Size	☐

Figure 5.6: IPerf server configuration on node 2


Experiment Preferences

Duration (s) 120

Automatically configure all Wireless Interfaces?

Figure 5.7: Experiment preferences

5.2.3 Application Initialization

Once applications are configured in the nodes, we need to trigger their initiation and decide which flow will have the experiment. Two options are available: either an overall duration of the experiment is specified (two minutes by default), or the user decides which group starts their applications. For this experiment, an overall duration is used. By clicking on the “Preferences” button, overall duration is changed, and optionally the user can choose if nodes are to be configured automatically (Figure 5.7).

Thanks to the OEDL DSL structure, an experiment definition is divided in blocks, which can be easily identified through the previously inserted input. For early adopters of this platform, observing the structure of the generated code and user input, can lead to a better understanding of what is inside of an experiment definition. This way the user can better learn the structure of an OEDL script, and easily adapt to its syntax.

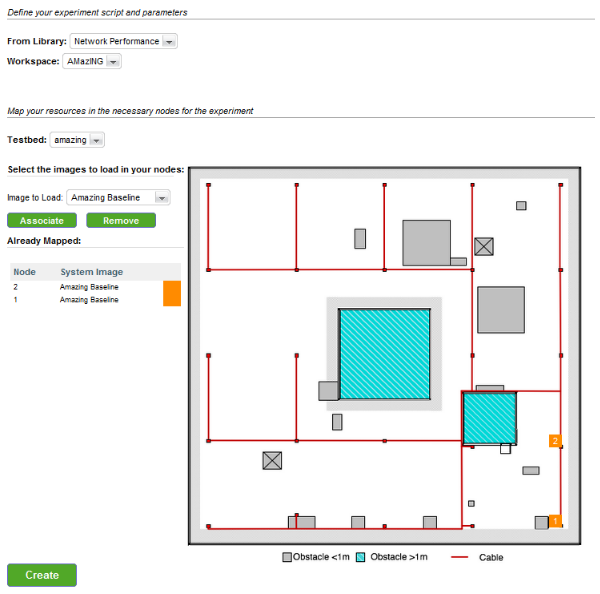
5.2.4 Revision Control

Some applications have time constraints, which are not dependent on the experiment overall duration. Both client and server were configured to live for thirty seconds. According to our experiment, this is not correct, since experiment duration is much higher (120 seconds), than the applications’ “internal” lifetime. Application configuration is as generic as possible, so the user must have knowledge of the tools running in nodes in order to configure them appropriately. Resource map and experiment definition can both be changed at any time or the user can branch from a certain revision. Experiment time is adjusted to `iperf` application duration.

5.2.5 Results

The experiment is created (Figure 5.8), in a workspace, called “AMazING”, proceeding to the resource map creation. As stated previously, system images are assigned to the experiment nodes. Once the user selects his experiment definition, it will be sent to the server to recognize which nodes are needed for the experiment. A client in node one and the server in node two is what was envisioned for this experiment. The used image is one of the baseline provided by the portal.

Next, an experiment run is queued, issuing either a single or multiple runs (previously referred to as batch run). An email is received once the experiment finishes, and we can either download the results or visualize the data using the provided graphics interface. The portal


Define your experiment script and parameters

From Library: Network Performance

Workspace: AMazING

Map your resources in the necessary nodes for the experiment

Testbed: amazing

Select the images to load in your nodes:

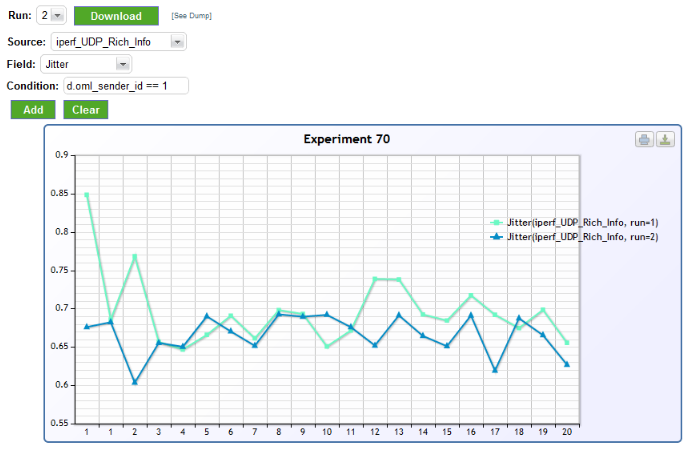
Image to Load: Amazing Baseline

Already Mapped:

Node	System Image
2	Amazing Baseline
1	Amazing Baseline

Figure 5.8: Create the experiment, by selecting the experiment definition and resources

provides users with means to generate graphics based on the measurement data. It does not provide a complete environment such as MATLAB, but allows basic graphics to be drawn, being able to make quick comparisons with the obtained measurements (Figure 5.9).


Run: 2

Source: iperf_UDP_Rich_Info

Field: Jitter

Condition: d.oml_sender_id == 1

Figure 5.9: Jitter comparison between two runs

`iperf` provides many measurements, for both TCP and UDP traffic analysis. This experiment uses UDP traffic. Using the graphics generation interface the user can observe both bandwidth and jitter values (Figure 5.9 and 5.10). As explained before, experiment measurement data comes in a SQLite database. Therefore, in this interface we choose the tables and fields to display, and a simple condition to filter results. The database also provides metadata related to groups and experiment properties. In the example, a filter is applied in the `oml_sender_id` property, which is a column in the metadata table that defines an id for each group of the experiment definition. This property indicates which group is generating the measurements. The Figure 5.10 shows graphics related to the node two, the receiver group. The measured bandwidth value is close to the inserted one, which was 3187.5 kBytes/sec.

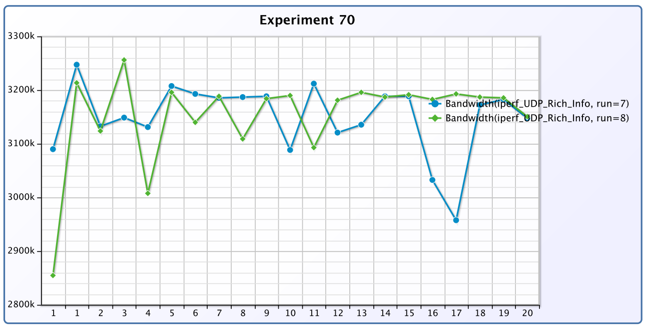


Figure 5.10: Bandwidth values comparison

More metrics are captured by the `iperf` application: Packet Loss Rate, Packets Loss, and Total Packets. The user can download the results, accessing all the measurement information for use in other applications such as MATLAB or SQLite Database Browser.

5.3 Experiment - Connectivity Monitor

In the previous scenario it was explained the basic use of the portal. It is important for a user to know the workflow behind experiment execution to understand how it better fits in a desired scenario. OEDL's biggest advantage is the automation and configurability gained through his language. Combined with the UI, this enables users to create experiments more easily, and to learn about their possibilities on an OMF enabled-testbed, by observing the experiment scripts. Furthermore, a multi-user experience is pervasive, sharing experiment results, software and experiments.

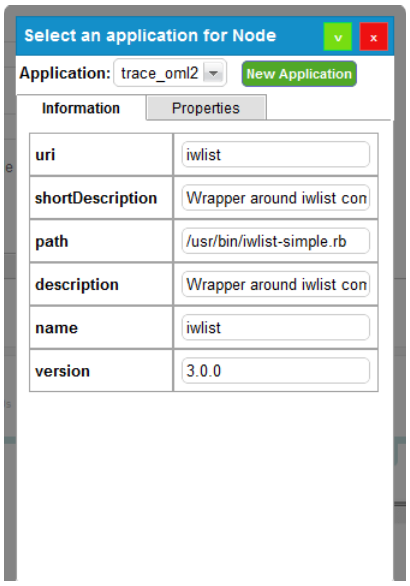
In this scenario, a more advanced experiment is explained. A wireless connectivity monitor is reproduced, which will measure link status between two nodes. It will capture the Received Signal Strength Indication (RSSI) values and nearby Wide Area Network (WAN) information (channel, MAC address, ESSID, etc). In this scenario, a sender, a receiver, and the `iwlist` are the applications used. The bundled OMF traffic generator and receiver are here used.

5.3.1 Application Creation

The same procedures are repeated as in the previous experiment: network configuration and applications configuration. The difference list in the selection of the application which will be OMF Traffic Generator (OTG) and OMF Traffic Receiver (OTR). OTG is deployed on node 1 and OTR in node2.

Groups are automatically created for each configured application. For wireless link connectivity `iwlist` is used, as supplied by the atheros wireless driver. It display statistics about the wireless interfaces, such as the RSSI, the channel being used and ESSID of the configured network. Integration of `wlanconfig` is installed by default in the baseline system image, useful when madwifi driver is used. The main benefits of this OML integration are in the ability to collect custom application metrics. Metrics will be collected in a database related to an experiment run, avoiding the usual log flooding when making tests manually. Moreover, this integration allows a user to easily control application lifetime and reuse it with another experiment.

As explained in section 3.1.7, a wrapper must be created in order to integrate OML within our applications. Two options are available: a native one (if your application is written in C/C++) or a binary wrapper. A native wrapper provides raw access to the whole application being able to inject metrics more easily. A binary wrapper injects measurements according to the application generated data: output text, log files. When developing third-party application wrappers, depending on its size, a binary wrapper may be preferable due to the lack of knowledge on the internal application logic. Yet the native provides access to more application data compared to the binary due to their dependency on the binary generated data.



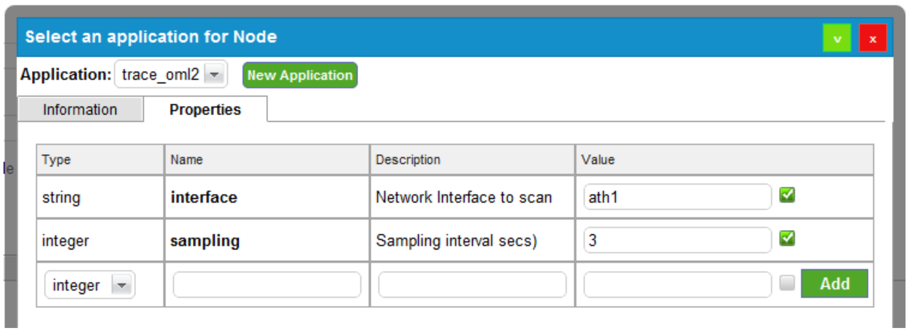
Application: trace_oml2

uri	iwlist
shortDescription	Wrapper around iwlist con
path	/usr/bin/iwlist-simple.rb
description	Wrapper around iwlist con
name	iwlist
version	3.0.0

Figure 5.11: Application Information

For the `iwlist`, a binary wrapper was developed. The creation of this wrapper implies that the user will get more familiar with OML libraries and the way to instrument an application with OEDL. A wrapper must come with an application definition that is responsible for the nodes' interaction during the experiment. The portal simplifies this task by generating the application definition for users, although with some limitations: the metrics must be added manually on the experiment script, and the developed application binary/wrapper must be included in the deployed system images. FTP access to system images is provided for users to maintain their application binaries up to date.

To use an application, users must first select the nodes where it will be deployed, followed by a push in "“New Application” . Displaying two forms, information about the application and options available must be filled. The information form (see Figure 5.11), includes version, name description, application path in the target node, etc. Application Properties (see Figure 5.12), corresponds to their command-line options, where a type and name must be indicated. Optionally, if the value field is checked and filled, the application is added to the nodes. The iwlist application will have two options: interface to listen and the interval between measurement injections. This application definition will be used in the experiment definition, being later translated by the EC to a CLI command to be executed on groups.



Select an application for Node

Application: trace_oml2 — New Application

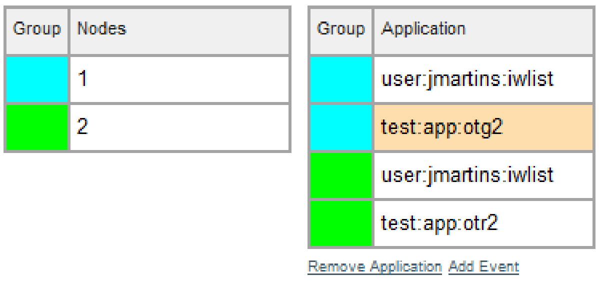
Information | Properties

Type	Name	Description	Value
string	**interface**	Network Interface to scan	ath1 ✓
integer	**sampling**	Sampling interval secs)	3 ✓
integer			Add

Figure 5.12: Application Properties

5.3.2 Timeline

Until now, the applications and nodes required for this experiment were configured (Figure 5.13). An event is triggered once applications and nodes are installed and ready to start. Experiment run logic will be enclosed within this event callback: the user is responsible for notifying the experiment's end; time semantics are given by wait system calls. Due to this kind of operation, a definition of timeline is better fitted to user action definition. Before the sapplications starts, users may want to load kernel modules or initiate a daemon supporting a network protocol, and later to trigger applications possibly with a specific synchronization.



Group	Nodes
🟦	1
🟩	2

Group	Application
🟦	user:jmartins:iwlist
🟦	test:app:otg2
🟩	user:jmartins:iwlist
🟩	test:app:otr2

Remove Application Add Event

Figure 5.13: Groups and applications configured

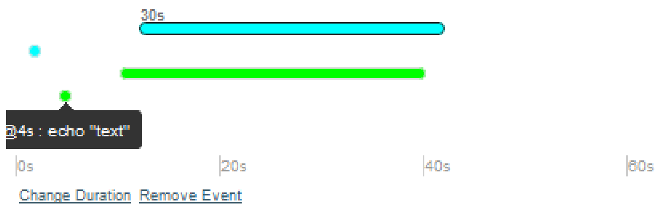


Figure 5.14: The long bars represent group applications and dots represent commands.

The timeline is the IEE UI element that allows users to choose when to initiate their applications and/or execute commands. Users choose the items from tables and choose a timestamp on the time scale to issue an action: if a group is selected a shell command is required; for applications selection, its execution duration is required. Figure 5.14 illustrates the timeline for this scenario. Two commands are executed on the nodes, before the applications start. `echo` UNIX command was used to exemplify this functionality. Dots represent these commands, and long bars represent applications duration during experiment execution. The receiver node group will be the first to start its applications and the sender group will start after two seconds. The portal will generate the corresponding source code for this experiment flow, both for the creation of the application and the execution flow for the experiment.

5.3.3 Results

The `iwlist` application will captures RSSI (both parameter and in dbm), essid, MAC address and channel values for wireless monitoring. In the first run, node 1 sends traffic to node 2 and both will measure connectivity. The second run has the exact same flow but without traffic generation/sink enabled. We can observe experiment results in Figure 5.15 and 5.16. Optionally, the user can download the results for later use.

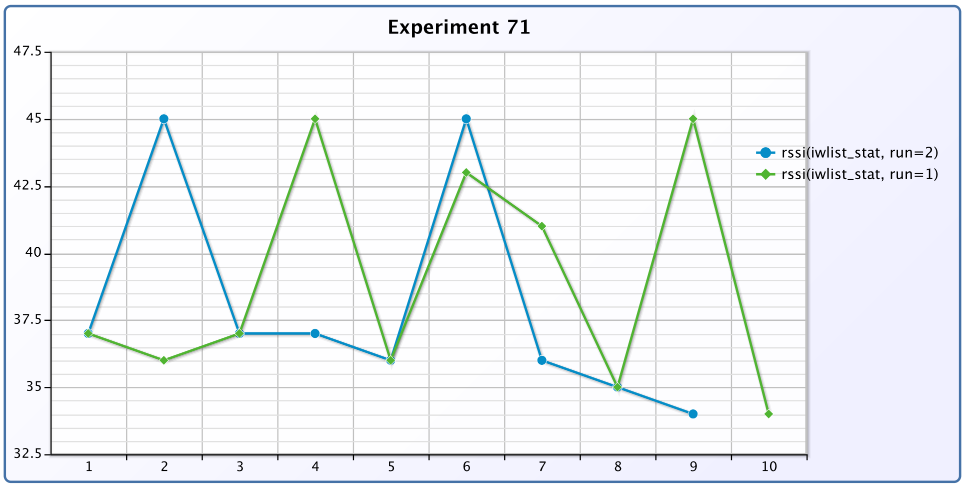


Figure 5.15: RSSI between two runs

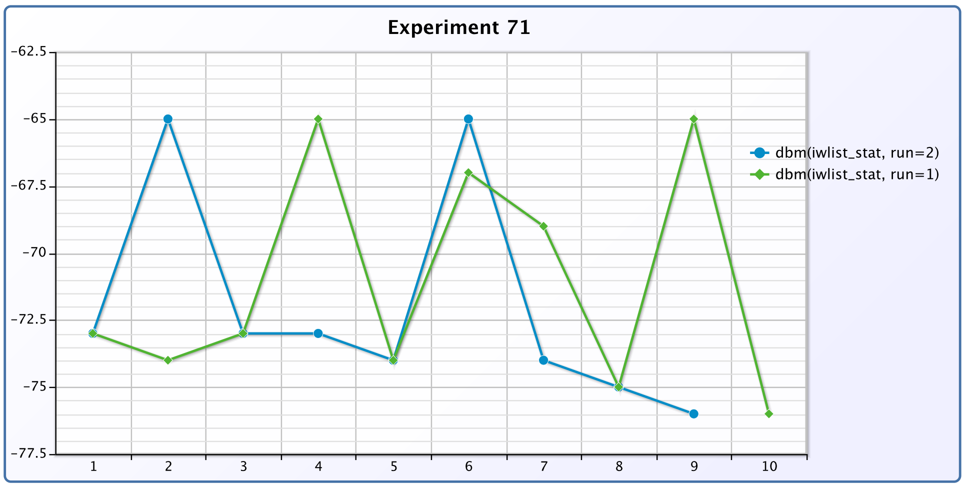


Figure 5.16: Power level (dbm)

5.4 Web Application Performance

The created web application conforms to typical standards, specifically HTML5. Its foundation lies upon the Rails web framework, which has various mechanisms to optimize page responsiveness:

- SQL Queries Caching: The main bottleneck in web applications is the Input/Output. The more requests are made to the web server, the more requests it has to handle, slowing each incoming request when the server is busy. From the server side, before the user gets the page, he must first query the database. This is a blocking operation, which introduces greater overhead. To solve this, Rails caches database operations from queries made to the database server.
- Cached Classes: Ruby is known for being a heavy language. Among the three different environments (test, development and production), rails will load up rails and application classes. In production mode, the rails environment caches all the classes' developed (controllers, models and libraries). This results in rails being less memory-hungry, and improves page responsiveness.
- Client-side Caching: Rails generates pages through their views. Taking advantage of browser cache capabilities, special headers are embedded in its views that tell the browser to cache page resources. Resources include images, JavaScript libraries, style sheets and the page content. Pages contain the `max-age` HTTP header to indicate the maximum time for which it will store a certain resource in the browser cache. Once that time passes, the resource is fetched again. Entity Tags (ETags)" is more related to data, where a set of data generates a unique hash, that becomes invalid once the data changes. This is more useful when a proxy mediates the resources between the server and multiple clients, referred to as reverse proxy cache.

Table 5.1: YSlow results for every page

`<model>#<action>`	score
workspaces#show	82 – B
workspaces#new	87 – B
workspaces#index	81 – B
workspaces#assign	83 - B
workspaces#users	82 - B
eds#show	79 – C
eds#new	75 – C
eds#index	83 – B
sysimages#show	83 - B
sysimages#index	82 – B
sysimages#new	85 – B
testbeds#show	84 – B
testbeds#index	87 – B
experiment#new	85 - B
experiment#show	75 - C
experiment#queue	87 - B

YSlow is a tool analyzes web pages based on a set of rules[83] for high performance websites. It comes in the form of a browser extension, and is very useful for testing performance of a web application (client side only). These tests were made for all displayable actions. Overall results were good, getting grade B in almost every action. The fact that some served files lacked ETags or Expire Headers to allow browser cache, and the non-existence of a Content Delivery Network to serve only static files, are the only issues with the web application performance. Furthermore, those actions displaying the text editor received lower grades due to its engine. This evaluation was important to evaluate web application construction, and to detect its flaws for future optimization.

Chapter 6

Conclusion

In the scope of this thesis, there were many items of research: to study existent testbed management platforms, to study their different usages and methodologies, and to identify the main requirements to serve better network testing needs, and scientific studies.

A platform was created to support this work allowing an experimenter to define network scenarios that are encapsulated in different network areas. The design of an experiment involves many steps and passes through an iterative process until a scenario is successfully described. With OMF deployed on AMazING testbed, most of this complexity is reduced, though only for a single user. OMF alone only presents means to instrument a testbed but not a complete system, and it is not prepared for the use of multiple users. This platform, unified with OMF framework, presents a testbed management system where experimentation is facilitated by hiding most of its complexity and enabling the user to focus more on the scenarios and results he needs to obtain. Several users can share the same platform; collaboration emerges as an important requirement for them to share their results and to reuse scenarios or even resources. Users can work together to create experiments, having version control to improve its development. This platform also introduces an environment to create experiments which helps them to adapt better to the whole platform, while allowing an increased focus on their scientific studies. Furthermore, it enables users to create experiments in a methodical and repeatable way, with version control and batch run mechanisms while being supplied with a usable interface to ease most of these tasks.

The identified requirements were fulfilled according to the objectives of this thesis, namely the study of testbed management systems and identification of different testbeds and their needs, improvements on the OMF platform to extend it to a testbed management system, and to provide users with a usable and iterative manner to create and describe their experiment scenarios.

6.1 Contributions

OMF is a state-of-art software, it is still in early stages and needs time to gain both stability and support from the majority of testbed deployments. The AMazING Panel has specific needs, as OMF sometimes fails in the integration with third party software, ranging from driver support, lack of information accessibility, bug fixes and minor inconsistencies.

Concerning this issues, several patches were contributed, to improve AMazING OMF deployment:

- Periodical image load logging - As previously mentioned, one of the OMF problems is the inability of accessing internal state of the EC when the experiment is running. Data such as properties being configured or progress of the experiment is only available at its end. AMazING OMF was patched to provide logging data for the AMazING panel. As images are loaded into nodes, XML data is now generated during this process, enabling the Portal to visualize experiment preparation progress.
- Experiment naming - As explained previously in the PubSub model, OMF creates XMPP nodes which powers experiment-related communication. Experiments are identified by their ID and created under a slice group. The naming upon its creation is automatically generated by OMF. Furthermore its related data is stored under files based on this naming. The portal must have control in experiments slice and ID to be able to access this data either for administration purposes (e.g. track experiment progress) or for users to have feedback of the results their experiments (e.g. log files, database files).
- CM integration - The AMaZING power control was integrated with OMF. It now allows access from the AM web services or from the experiments scripts.
- `ath5k` drivers support - `ath5k` drivers are more stable and maintained than the madwifi drivers. A driver were added to the RC codebase.
- `iwlist` application support - OMF has support for the wlanconfig application to access wireless network information with devices using madwifi driver. Since AMazING testbed uses Atheros drivers, no applications are integrated with the testbed to access this information. A wrapper was developed, following the OML integration process (see section section 3.1.7), allowing users to access this information.
- `ath9k` driver physical device name - OMF added support for `ath9k` driver, although lacking on consistence in the physical device naming between the kernel and OMF. In the Atheros devices this physical device is changed every time the kernel module is loaded. An installation of the RC must come with a manual association between OMF resource path and physical device name. This problem, led to an invalid state of RC that would affect future experiment runs. RC was modified, accessing kernel device naming and creating this association dynamically.
- Available devices discovery - As the previously mentioned association could lead RC to an invalid state, an automatic discovery mechanism between the available network interfaces and OMF was introduced, bringing more stability to nodes. Varying from the loaded driver, it will search the system for the appropriate interface name.

6.2 Future Work

Testbed management systems are almost non-existent, proving this solution to be a relevant initiative. This application is still in its early stages, with room for more improvements and new features. There are processes that can be improved from the experimenter side and new functionalities to allow testbed owners to make more administrative tasks. In addition, OMF is making its way to support PlanetLab in the next versions, so this could inflict general changes in the experiment creation workflow, since access to external resources would be allowed. Furthermore, System images creation still is a low-level process where a build system similar to the used by OpenSuse[84] could also be implemented. Moreover, the IEE UI may need enhancements and user workflow processes could be optimized in some ways, like merging resources maps into the IEE. Timeline could display more information while events could allow more actions (e.g. power control of nodes). OEDL capabilities are partially implemented. Advanced OEDL usage, such as topologies to enforce link in nodes or dynamic properties, is not implemented in the IEE. Some issues related to its stability were found, being used only a reduced set of routines of the OEDL. The implementation of such advanced features must be asserted with the OEDL available support.

The OMF still has some scalability problems supporting different types of network interfaces (3G, ZigBee, Bluetooth, etc.) and wireless card drivers. Scheduling mechanisms currently deployed are not concurrent due to experiment results correctness. Other different approaches should be evaluated to assert the possibility of improving the availability and throughput of the testbed to execute experiments concurrently. On a related subject, the testbed is not yet ready for mobility scenarios, due to the support of both hardware and software infrastructure to describe movement on the testbed. This introduces a new set of test scenarios that are useful in areas such as vehicular networks. The platform still needs a full-fledged environment to analyze results, or ways to document the experiment scenarios, such as the notebook feature offered by OMF Portal.

NEPI framework is a great initiative and some ideas could be integrated, providing users the use of different backbends to describe bigger scenarios. One of AMazINGs testbed objectives is to integrate NS-3 into testbed infrastructure. Experiments could be made through NS-3 Python scripts, extending the testbed, while inviting simulation adopters to use this testbed.

Appendix A

Experiment Definitions

In OMF-enabled testbeds, experiment workflows are described through an OEDL script, namely the experiment definition. This appendix examplifies its use in the Network Performance experiment (section 5.2). The group definition and resource configuration are defined as explained in section 3.1.3. The applications configured on groups are described according to the experiment described in section 5.2.

Experiment 1 Network Performance

```
defGroup("__group_n1_", "omf.amazing.node1") do |node|
  node.addApplication("test:app:iperf") do |app|
    app.setProperty("udp", true)
    app.setProperty("port", 5000)
    app.setProperty("time", 20)
    app.setProperty("bandwidth", 25500000)
    app.setProperty("client", "192.168.0.2")
    app.measure("Peer_Info", :samples => 1)
    app.measure("UDP_Periodic_Info", :samples => 1)
    app.measure("UDP_Rich_Info", :samples => 1)
  end
  node.net.w1.mode = "ad-hoc"
  node.net.w1.essid = "test"
  node.net.w1.type = "g"
  node.net.w1.channel = "6"
  node.net.w1.ip = "192.168.0.1"
end
```

```
defGroup("__group_n2_", "omf.amazing.node2") do |node|
  node.addApplication("test:app:iperf") do |app|
    app.setProperty("udp", true)
    app.setProperty("port", 5000)
    app.setProperty("time", 20)
    app.setProperty("server", true)
    app.setProperty("interval", 1)
    app.measure("Peer_Info", :samples => 1)
    app.measure("UDP_Rich_Info", :samples => 1)
  end
  node.net.w1.mode = "ad-hoc"
  node.net.w1.essid = "test"
  node.net.w1.type = "g"
  node.net.w1.channel = "6"
  node.net.w1.ip = "192.168.0.2"
end
onEvent(:ALL_UP_AND_INSTALLED) do |node|
  wait(30)
  allGroups.net.w0.down
  allGroups.startApplications
  wait(30)
  allGroups.stopApplications
  Experiment.done
end
```

Below it is illustrated the definition used in the Connectivity Monitor experiment (section 5.2). The resource and application configuration takes the same steps as the previous experiment with the custom application added to monitor the wireless connectivity, namely the `iwlist`.

Experiment 2 Connectivity Monitor

```
defGroup("__group_n1_", "omf.amazing.node1") do |node|
  node.addApplication("user:jmartins:iwlist") do |app|
    app.setProperty("interface", "ath1")
    app.setProperty("sampling", 3)
    app.measure("stat", :samples => 1)
  end
  node.net.w1.mode = "ad-hoc"
  node.net.w1.essid = "network"
  node.net.w1.type = "g"
  node.net.w1.channel = "6"
  node.net.w1.ip = "192.168.0.1"
end
defGroup("__group_n2_", "omf.amazing.node2") do |node|
  node.addApplication("user:jmartins:iwlist") do |app|
    app.setProperty("interface", "ath1")
    app.setProperty("sampling", 3)
    app.measure("stat", :samples => 1)
  end
  node.net.w1.mode = "ad-hoc"
  node.net.w1.essid = "network"
  node.net.w1.type = "g"
  node.net.w1.channel = "6"
  node.net.w1.ip = "192.168.0.2"
end
onEvent(:ALL_UP_AND_INSTALLED) do |node|
  wait(60)
  wait(1)
  group("__group_n1_").exec("echo ẗext")
  wait(3)
  group("__group_n2_").exec("echo ẗext")
  wait(6)
  group("__group_n2_").startApplications
  wait(2)
  group("__group_n1_").startApplications
  wait(28)
  group("__group_n2_").stopApplications
  wait(2)
  group("__group_n1_").stopApplications
  Experiment.done
end
```

Bibliography

[1] "*The ns-2 manual*," March 2010. [Online]. Available: http://www.isi.edu/nsnam/ns/doc/

[2] "*ns-3*," March 2010. [Online]. Available: http://www.nsnam.org/

[3] "*OMNET++*," April 2010. [Online]. Available: http://www.omnetpp.org/

[4] "*INET component package*," April 2010. [Online]. Available: http://inet.omnetpp.org/

[5] "Java in Simulation Time / Scalable Wireless Ad hoc Network Simulator," April 2010. [Online]. Available: http://jist.ece.cornell.edu/

[6] X. Zeng, R. Bagrodia, and M. Gerla, "GloMoSim: a library for parallel simulation of large-scale wireless networks," *Proceedings. Twelfth Workshop on Parallel and Distributed Simulation PADS '98 (Cat. No.98TB100233)*, pp. 154–161, 1998.

[7] "PARSEC Parallel Simulation Software," April 2010. [Online]. Available: http://pcl.cs.ucla.edu/projects/parsec/manual.pdf

[8] "A Discrete-Event Simulation Course Based on the SimPy Language," April 2010. [Online]. Available: http://heather.cs.ucdavis.edu/~matloff/simcourse.html

[9] G. F. Riley, "The Georgia Tech Network Simulator," *Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research - MoMeTools '03*, no. August, p. 5, 2003.

[10] G. Riley, "Large-scale network simulations with GTNetS," *Simulation Conference, 2003. Proceedings of the 2003 Winter*, vol. 1, pp. 676–684 Vol.1, 2003.

[11] "*QualNet Developer*," April 2010. [Online]. Available: http://www.scalable-networks.com/products/qualnet/

[12] D. T. S. P. A. Farrington, H. B. Nembhard and G. W. Evans, "Network Simulations with OPNET," *Proceedings of the 1999 Winter Simulation Conference*, pp. 307–314, 1999.

[13] "OPNET Modeler," May 2010. [Online]. Available: http://www.opnet.com/solutions/network_rd/modeler.html

[14] B. White, S. Guruprasad, M. Newbold, J. Lepreau, L. Stoller, R. Ricci, C. Barb, M. Hibler, and A. Joglekar, "Netbed : An Integrated Experimental Environment," vol. 32, no. 3, pp. 2002–2002, 2002.

[15] 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,” *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, p. 255, Dec. 2002.

[16] K. Fall, “Network emulation in the vint/ns simulator,” *Proceedings IEEE International Symposium on Computers and Communications Cat NoPR00250*, pp. 244–250, 1999.

[17] L. Rizzo, “Dummynet: a simple approach to the evaluation of network protocols,” *ACM Computer Communication Review*, vol. 27, no. 1, pp. 31–41, 1997.

[18] M. Hibler, L. Stoller, J. Lepreau, R. Ricci, and C. Barb, “Fast , scalable disk imaging with frisbee,” *Image Rochester NY*, pp. 283–296, 2003.

[19] E. Eide, “An experimentation workbench for replayable networking research,” *Symposium A Quarterly Journal In Modern Foreign Literatures*, pp. 215–228, 2007.

[20] C. K. I. S. Y. Miyachi T, “Starbed and springos: Large-scale general purpose network testbed and supporting software,” *ACM International Conference Proceeding Series*, vol. 180, 2006.

[21] K. Yocum, K. Walsh, A. Vahdat, P. Mahadevan, D. Kostic, J. Chase, and D. Becker, “Scalability and accuracy in a large-scale network emulator,” *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 3, pp. 28–28, July 2002.

[22] S. Hemminger, “Network emulation with netem,” *Linux Conf Au*, no. April, 2005.

[23] S. Kurkowski, T. Camp, and M. Colagrosso, “Manet simulation studies: the incredibles,” *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 9, no. 4, pp. 50–61, 2005.

[24] J. Bicket, D. Aguayo, S. Biswas, and R. Morris, “Architecture and evaluation of an unplanned 802.11b mesh network,” *Proceedings of the 11th annual international conference on Mobile computing and networking - MobiCom ’05*, p. 31, 2005.

[25] A. Raniwala and T. Chiueh, *Architecture and algorithms for an IEEE 802.11-based multichannel wireless mesh network.* IEEE, 2005, vol. 3, pp. 2223–2234.

[26] S. ElRakabawy, S. Frohn, and C. Lindemann, “Scalemesh: A scalable dual-radio wireless mesh testbed,” in *Sensor, Mesh and Ad Hoc Communications and Networks Workshops, 2008. SECON Workshops ’08. 5th IEEE Annual Communications Society Conference on*, june 2008, pp. 1 –6.

[27] S. L. Shrestha, J. Lee, A. Lee, K. Lee, J. Lee, and S. Chong, “An Open Wireless Mesh Testbed Architecture with Data Collection and Software Distribution Platform,” *2007 3rd International Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities*, pp. 1–10, 2007.

[28] “Ultra high-speed Mobile Information and Communication Testbed,” April 2011. [Online]. Available: http://umic-mesh.net/testbed/realisation/

[29] C. Samsel, “ flowgrind ,” March 2011. [Online]. Available: http://www.ohloh.net/p/flowgrind

[30] H. Lundgren, D. Lundberg, J. Nielsen, E. Nordstrom, and C. Tschudin, "A large-scale testbed for reproducible ad hoc protocol evaluations," *2002 IEEE Wireless Communications and Networking Conference Record. WCNC 2002 (Cat. No.02TH8609)*, vol. 00, no. c, pp. 412–418, 2002.

[31] K. Ramachandran, K. Almeroth, and E. B. Royer, "A framework for the management of large-scale wireless network testbeds," *Proceedings of the First Workshop on Wireless Network Measurements WiNMee 2005*, 2005.

[32] P. De, a. Raniwala, S. Sharma, and T. Chueh, "MiNT: a miniaturized network testbed for mobile wireless research," *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, pp. 2731–2742, 2005.

[33] " LEGO Mindstorms ," March 2011. [Online]. Available: http://mindstorms.lego.com/

[34] P. De, A. Raniwala, R. Krishnan, K. Tatavarthi, J. Modi, N. A. Syed, S. Sharma, and T.-c. Chiueh, "Mint-m: an autonomous mobile wireless experimentation platform," in *Proceedings of the 4th international conference on Mobile systems, applications and services*, ser. MobiSys '06. New York, NY, USA: ACM, 2006, pp. 124–137.

[35] iRobot Corporation, " Robots that make a difference ," May 2011. [Online]. Available: http://www.irobot.com/

[36] MikroTik, " RouterBoard ," July 2011. [Online]. Available: http://www.routerboard.com/

[37] "Nam: Network Animator," June 2010. [Online]. Available: http://www.isi.edu/nsnam/nam/

[38] C. Mitchell, V. P. Munishwar, S. Singh, K. Gopalan, and N. B. Abu-Ghazaleh, "Testbed design and localization in MiNT-2: A miniaturized robotic platform for wireless protocol development and emulation," *2009 First International Communication Systems and Networks and Workshops*, pp. 1–10, Jan. 2009.

[39] 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*, vol. 15, no. 3, pp. 315–329, 2010.

[40] M. Ott, I. Seskar, R. Siraccusa, and M. Singh, "ORBIT Testbed Software Architecture: Supporting Experiments as a Service," *First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*, pp. 136–145, 2005.

[41] "World current OMF Deployments," April 2010. [Online]. Available: http://omf.mytestbed.net/wiki/omf/DeploymentSite

[42] A.-C. Anadiotis, A. Apostolaras, D. Syrivelis, T. Korakis, L. Tassiulas, L. Rodriguez, I. Seskar, and M. Ott, "Towards maximizing wireless testbed utilization using spectrum slicing," *Computer Engineering*, pp. 1–16, 2010.

[43] A.-c. A. Apostolos and A. Dimitris, "A new slicing scheme for efficient use of wireless testbeds," *Spectrum*, pp. 83–84, 2009.

[44] M. Portoles-Comeras, M. Requena-Esteso, J. Mangues-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 *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2006. TRIDENTCOM 2006. 2nd International Conference on*, 0-0 2006, pp. 10 pp. –266.

[45] P. Dini, M. Portoles-Comeras, J. Nin-Guerrero, J. Mangues-Bafalluy, L. L. Dai, and S. Addepalli, “A reconfigurable test platform to experiment with wireless heterogeneous networks in a laboratory,” *Int. J. Commun. Netw. Distrib. Syst.*, vol. 5, pp. 46–66, July 2010.

[46] “PlanetLab Architecture: An Overview,” March 2006. [Online]. Available: http://www.planet-lab.org/files/pdn/PDN-06-031/pdn-06-031.pdf

[47] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, “Planetlab: an overlay testbed for broad-coverage services,” *ACM SIGCOMM*, vol. 33, no. 3, pp. 3–12, 2003.

[48] “The OneLab Project,” March 2010. [Online]. Available: http://onelab.eu/

[49] T. Magedanz, F. Schreiner, and S. Wahle, “Service-oriented testbed infrastructures and cross-domain federation for Future Internet research,” *2009 IFIP/IEEE International Symposium on Integrated Network Management-Workshops*, pp. 101–106, June 2009.

[50] “Wireless EXperimentation Tool,” May 2010. [Online]. Available: http://planete.inria.fr/Software/Wextool/

[51] D. Dujovne, T. Turletti, and W. Dabbous, “Experimental Methodology For Wireless Networks,” INRIA, Research Report RR-6667, 2008.

[52] M. Lacage and M. Ferrari, “Nepi : Using independent simulators , emulators , and testbeds for easy experimentation,” *Workshop on Real Overlays and Distributed Systems ROADS*, 2009.

[53] D. Gomes, A. Matos, E. Fonseca, and R. Aguiar, “Deploying and testing a ngn testbed ist-daidalos testbed,” in *Testbeds and Research Infrastructures for the Development of Networks Communities and Workshops, 2009. TridentCom 2009. 5th International Conference on*, april 2009, pp. 1 –6.

[54] T. Rakotoarivelo, M. Ott, G. Jourjon, and I. Seskar, “Omf: a control and management framework for networking testbeds,” *SIGOPS Oper. Syst. Rev.*, vol. 43, pp. 54–59, January 2010. [Online]. Available: http://doi.acm.org/10.1145/1713254.1713267

[55] “The Ruby Language,” May 2011. [Online]. Available: http://www.ruby-lang.org/

[56] “ IPv4 Packet Filtering ,” March 2011. [Online]. Available: http://iptables.sourceforge.net/

[57] “ Ethernet Bridge Tables Admnistration ,” March 2011. [Online]. Available: http://ebtables.sourceforge.net/

[58] “The XMPP Protocol,” April 2010. [Online]. Available: http://xmpp.org/

[59] "RFC3920 - XMPP Core," April 2010. [Online]. Available: http://xmpp.org/rfcs/rfc3920.html

[60] "XMPP Protocol Extensions," April 2010. [Online]. Available: http://xmpp.org/xmpp-protocols/xmpp-extensions/

[61] "XEP-0060: Publish-Subscribe," April 2010. [Online]. Available: http://xmpp.org/extensions/xep-0060.html

[62] M. Singh, M. Ott, I. Seskar, and P. Kamat, "ORBIT Measurements Framework and Library (OML): Motivations, Design, Implementation, and Features," *First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*, pp. 146–152, 2005.

[63] J. White, G. Jourjon, T. Rakotoarivelo, and M. Ott, "Measurement architectures for network experiments with disconnected mobile nodes," *Network*, vol. 46, pp. 350–365, 2010.

[64] G. Jourjon, T. Rakotoarivelo, and M. Ott, "A portal to support rigorous experimental methodology in networking research," in *5th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities and Workshops*, Shanghai/China, April 2011, p. 16.

[65] "A Project Management platform," May 2010. [Online]. Available: http://www.redmine.org/

[66] "Git - Fast Version Control System," May 2010. [Online]. Available: http://git-scm.com/

[67] "The R Project for Statistical Computing," June 2010. [Online]. Available: http://www.r-project.org/

[68] J. a. P. Barraca, D. Gomes, and R. L. Aguiar, "AMazING – Advanced Mobile wIreless playGrouNd," in *6th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*. LNICST, 2010, pp. 219–230.

[69] M. P. De Leon, F. C. Grant, M. Roddy, M. G. Moreno, A. R. Vicente, and C. Jedrzejek, "Daidalos Framework for Successful Testbed Integration," *2007 3rd International Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities*, pp. 1–6, 2007.

[70] "The AMazING testbed Documentation," April 2010. [Online]. Available: https://helios.av.it.pt/projects/amazing/wiki

[71] "Specification of the IP Flow Information Export Protocol for the Exchange of IP Traffic Flow Information," June 2011. [Online]. Available: http://tools.ietf.org/html/rfc5101

[72] "Fonera power control node," April 2010. [Online]. Available: http://mytestbed.net/projects/omf/wiki/Planning-5-4

[73] T. Carrão, "Plataforma de medida e caracterização de redes sem fios," December 2010.

[74] "nProbe v6," April 2010. [Online]. Available: http://www.ntop.org/nProbe.html

[75] "The HTML5 Web Platform," June 2011. [Online]. Available: http://platform.html5.org/

[76] "Ruby on Rails," June 2011. [Online]. Available: http://rubyonrails.org/

[77] A. Leff and J. T. Rayfield, "Web-Application Development Using the Model/View/-Controller Design Pattern," *Enterprise Distributed Object Computing Conference, IEEE International*, vol. 0, p. 0118, June 2001.

[78] "Secure FTP made easy!" February 2011. [Online]. Available: http://www.pureftpd.org/project/pure-ftpd

[79] H. K. Wright and D. E. Perry, "Subversion 1.5: A case study in open source release mismanagement," in *Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, ser. FLOSS '09. Washington, DC, USA: IEEE Computer Society, May 2009, pp. 13–18.

[80] T. Swicegood, *Pragmatic Version Control Using Git.* Pragmatic Bookshelf, May 2008.

[81] D. Maurer, "Usability for Rich Internet Applications," June 2011. [Online]. Available: http://www.digital-web.com/articles/usability_for_rich_internet_applications/

[82] " Iperf ," February 2011. [Online]. Available: http://iperf.sourceforge.net/

[83] Y. D. Network, "Best Practices for Speeding Up Your Web Site," May 2011. [Online]. Available: http://developer.yahoo.com/performance/rules.html

[84] "OpenSuse Build System," May 2011. [Online]. Available: https://build.opensuse.org/