Enabling Iterative Development and Reproducible Evaluation of Network Protocols

Young-Hwan Kim, Alina Quereilhac, Mohamed Amine Larabi, Julien Tribino, Thierry Parmentelat, Thierry Turletti, Walid Dabbous

Enabling Iterative Development and Reproducible Evaluation of Network Protocols

Young-Hwan Kim*, Alina Quereilhac, Mohamed Amine Larabi, Julien Tribino, Thierry Parmentelat, Thierry Turletti, Walid Dabbous

INRIA, 2004 route des Lucioles, B.P. 93, 06902 Sophia Antipolis, France

Abstract

Over the last two decades several efforts have been made to provide adequate experimental environments, aiming to ease the development of new network protocols and applications. These environments range from network simulators providing a highly controllable evaluation conditions, to live testbeds providing realistic evaluation environment. While these different approaches foster network development in different ways, there is no simple way to gradually transit from one to another, or to combine their strengths to suit particular evaluation needs. We believe that enabling a gradual transition from a pure simulated environment to a pure realistic one, where the researcher can decide which aspects of the environment are realistic and which are controllable, allows improving network solutions by simplifying the problem analysis and resolution.

In this paper, we propose a new network experimentation framework where simulated and real components can be arbitrarily combined to build custom test environments, allowing refining and improving new protocols and applications implementations by gradually increasing the level of realism of the evaluation environment. Moreover, we present a testbed architecture specifically adapted to support the proposed concept, and discuss the design choices we made based on our previous experience in the area of network testbeds. These choices address key issues in network testbed development, such as ease of experimentation, experiment reproducibility, and testbed federation, to enable scaling the size of experiments beyond what a single testbed

*Corresponding author

Email address: young-hwan@inria.fr (Young-Hwan Kim)

would allow.

Keywords: Reproducible research, Network experiment, Iterative development, Simulation, Emulation, Network testbeds

---

1. Introduction

Today, network researchers proposing new protocols and services use many different environments to evaluate their solutions before in the wild deployment. These environments can be generally classified into mathematical modeling, simulation, emulation and live experimentation on real testbeds. Evaluating an operational solution involves tests covering the different parts of the protocol stack starting from the application, then the transport and network protocols downto the link and physical layers.

It is common to begin the evaluation of a new protocol using mathematical modeling or simulation, and as the protocol design is refined, to switch to more realistic evaluation environment, e.g., first with emulation (e.g. using real application, protocol and OS code and simulated channel model), and later testbed experimentation with real nodes and real traffic.

This staged evaluation process is *costly* in terms of code development (e.g., we might be obliged to re-implement the same protocol twice, once in the simulator and once in the Linux kernel or user space) and usually results in *inaccurate evaluation* (comparison of different algorithms in a simulator and in a testbed might not yield the same results). There is however a need to easily reach a deployable solution whose performance is fairly accurate. Emulab [1] tackles the problem of code reimplementation by using the same scripting language for simulation and emulation, allowing an easy transition from simulation to emulation and vice-versa. However, none of existing solutions, including Emulab, provide a fine grained control on the realism level of the evaluation environment.

In this paper, we propose a network experimentation methodology named IDEV for Iterative Development of Network Protocols. IDEV empowers researchers to choose which parts of an experiments should be simulated. Our methodology supports an easy switch from full simulation (all parts including application, protocols, and link layers are simulated) to live experimentation on real nodes where simulations are not used.

In this work, we focus on the evaluation of wireless network protocols to illustrate the complexity of combining simulation and real experimentation.

Indeed, the characteristics of wireless links are known to be very variable, unpredictable, and hardly controllable. So, reproducibility of wireless experimentation results is only possible using complex and costly testbeds such as a Faraday Cage shielding Radio Frequency (RF) interference from the outside world and an anechoic chamber to prevent radio waves reflections on the walls.

We propose a set of tools and a testbed architecture to implement the IDEV concept. The proposed framework and testbed leverage on existing technologies, such as the ns-3 network simulator [2], and Direct Code Execution (DCE) [3] for simulation and emulation support, as well as on the OMF wireless framework [4] to support experimentation on real hardware. Our testbed includes 40 high performance wireless nodes and an RF anechoic chamber embedded into a Faraday cage to enable reproducible wireless experimentation results.

The proposed architecture also makes use of the Network Programming Interface (NEPI) [5], to simplify the description, execution and control of the same experiment scenario over different environments. NEPI provides a high-level API to not only interact with resources from different evaluation environments, such as ns-3 and OMF, but also automate network experiment steps including experiment set-up, application installation, and collection of results.

The rest of the paper is organized as follows. In Section 2, we discuss the related work and their differences with out approach. In Section 3, we describe in detail the IDEV methodology and provide several use cases. In Section 4, we present the testbed architecture and infrastructure and we draw conclusions in Section 5.

2. Related work

In the network research community, reproducibility is becoming more popular, and some emulation based tools are available to reproduce experimental results. Handigol et al. [6], replayed thirteen network experiments using Mininet, a container-based network emulator. Mininet enables running standard Unix/Linux network applications and real Linux kernels, in a lightweight and inexpensive way. However, this tool does not allow easily transforming a simulation into an emulation. For example, it is not possible to validate a simulated network module with links emulated in Mininet. Indeed, the gap between simulating a network protocol and emulating it is

usually high. Meanwhile, some approaches have been proposed that mix the two worlds: ns-3 [2], Emulab [1], and Flexlab [7].

Ns-3 supports not only simulation, but also emulation using real network links, real applications and network protocol stacks [8, 3]. Thus, an experiment using ns-3 can easily be ported between simulation and emulation, by slightly modifying the scenario script.

Emulab uses the ns-2 scripting format to compose network scenarios, thereby it makes it easy to reuse the same script in the Emulab testbed and the ns-2 simulator. However, this tool cannot provide realistic and reliable network conditions. In the Emulab testbed, wired nodes are connected through Ethernet links, but the channel environment is configured and controlled by pre-determined or stochastic channel models, thus the results depend on the reliability of the models. Moreover, wireless nodes are placed in an open space and are exposed to radio frequency (RF) noise and interference. Therefore, Emulab is not adequate for reproducible research requiring realistic network conditions.

Flexlab is an hybrid testbed combining some strengths and weaknesses from Emulab and PlanetLab [9]. Although Flexlab is not supported by Emulab since 2007, the approach is remarkable. As mentioned above, Emulab provides full controllability, but the network links are artificially emulated. On the other hand, network links in a real testbed (e.g., PlanetLab) are time-varying as they are connected through the wild Internet. Flexlab provides a portal within the Emulab management system enabling interconnection with PlanetLab nodes. Through the portal, Flexlab can measure network characteristics on real PlanetLab links, and can configure them accordingly into Emulab.

To summarize, none of these approaches allows varying the level of experiment realism in an easy and fine granularity, and this is what IDEV aims to offer.

3. Iterative Development (IDEV) of Network Protocols

Network simulation predicts the outcome of real experiments by interactions between the different network entities, such as nodes and protocol stacks. In general, they provide various configurable attributes to evaluate how the network behaves under different conditions. Simulators provide higher flexibility than mathematical models, imitating the behavior of real systems, while still enabling high controllability [10]. However, simulators

provide a simplified view of the rules that govern the iteration between network entities, and so the realism that they achieve is limited.

Network emulation combines realism and controllable modelization, using simulation and real software and/or hardware, but it does it in a limited and specific way. Emulation is often used for performance evaluation, such as to predict the impact of changes in existing protocols and applications or to optimize technological decision-making. There are three different types of network emulation, depending on which components are real and which components are modeled: application-level, protocol-level, and link-level. The *application-level* emulation is the most frequently used, and it replaces deterministic or stochastic traffic generated by live applications or replaying previously captured traffic traces. *Protocol-level* emulation uses real protocol stacks instead of simulated ones. *Link-level* emulation connects nodes by real wired or wireless links. Finally, experimentation on real testbeds involves not only hardware and software components (e.g., applications, protocol stacks, and network interfaces), but also environmental conditions similar to those of the final deployment environment.

Ideally, researchers should be able to conduct both realistic and controlled experiments [11], choosing when simulation, emulation or real hardware is best suited for their study. However, this is not trivial to achieve and no standard support for such an evaluation approach exists today.

In this paper, we propose the Iterative Development of Network Protocols (IDEV) methodology to combine simulation, application-level emulation, protocol-level emulation, link-level emulation, and real experimentation in a single test environment. This should be done in a very fine granular level, so as to enable using more than one technique for a same network component (e.g., a network node could have simulated, emulated and real components at the same time).

In the following, we describe the IDEV methodology through use cases and explain the different components of this framework.

3.1. Framework Components

The IDEV framework provides simulated, emulated, and real components as constructive blocks to build custom network test environments. Figure 1 depicts the basic idea of the framework, where the applications, the network protocol stack, and the network infrastructure can be independently specified by the user, to be either simulated, emulated or implemented in real.

IDEV uses the components described in the following to create fully simulated environments, real environments, and many intermediate emulation environments involving both simulated and real components.

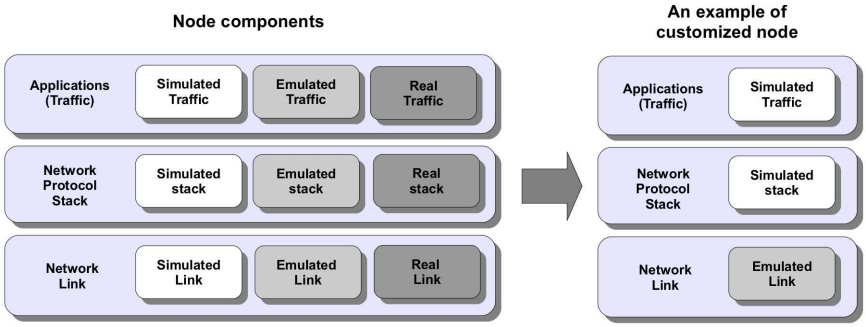


Figure 1: Simulated, emulated, and real constructive blocks are used to create custom test environments.

3.1.1. Simulation components

Simulated components leverage on the ns-3 network simulator, a discrete event network simulator that provides fairly accurate models for wireless devices and channels. The ability of ns-3 to generate simulated traffic can be exploited to integrate ns-3 simulations with real networks. These features make ns-3 a very versatile test environment, and thus a key component for our framework.

3.1.2. Emulation components

To provide emulation, we take advantage of a recent extension to the ns-3 simulator, DCE [3], which allows to run unmodified C/C++ based applications and protocols on top of ns-3. DCE re-implements a subset of the standard socket APIs (glibc) used by the simulated applications to override specific system calls with ns-3 compatible operations. It enables to use both user-space and kernel-space protocol implementations inside a same ns-3 simulated node. DCE implements sophisticated techniques to isolate global and static variables on multiple instances of a same protocol implementation in a single process. In precise, DCE can use multiple nodes in the same execution environment to execute the same unmodified protocol implementation

without interfering with each other. So, DCE allows us to add emulation capabilities to the IDEV framework at both application and network protocol stack levels.

The ns-3 EmuNetDevice (EMU) [8] further increases the emulation capabilities of the framework, by allowing to transparently attach a ns-3 simulated node to a physical wireless network interface, and so to a real wireless channel. EMU is an hybrid ns-3 device, which is seen as any other simulated device from within the simulated network. More precisely, EMU opens a raw socket in promiscuous mode to a physical network interface in the host machine. If the EMU device is correctly configured with a different IP address than the physical interface it is attached to, it can distinguish the traffic addressed to its own IP address, and inject the distinguished traffic to the simulated network. Because the destination IP address of the traffic directed to the EMU device is different from the IP address of the physical interface, the latter will ignore this traffic, making possible the coexistence of simulated and real nodes.

3.1.3. Real components

The IDEV testbed enables physical nodes to use real applications, operating systems, and network interfaces. IDEV can be used to experiment with any Linux compatible application in the testbed because all nodes use the Linux operating system. Furthermore, the testbed allows to deploy and evaluate customized Linux images including user-modified kernel modules, system libraries, and instrumented applications on the nodes. All nodes are equipped with Ethernet and WiFi NICs. Further details on the physical characteristics of the testbed are provided in Section 4.

3.2. IDEV with ns-3

The proposed framework aims to become a general solution for

IDEV aims to provide a general solution for building custom network evaluation environments that mixes simulation, emulation and realistic components. In order to support seamless integration between these different types of components it is important to support realtime simulation and to use a common packet representation.

Realtime simulation is needed to synchronize simulated components with real hardware, such as real routers and Ethernet links. The ns-3 simulator not only supports realtime simulation, but also generates real packets that can then be seamlessly injected into a real network device. Ns-3 supports a

common packet representation format *pcap* [12] to capture and store packets in a compatible way with real network devices and protocol stacks. Through EMU and DCE, ns-3 simulations can include real network-links, applications and protocol stacks.

Figure 2 illustrates two use cases integrating simulated, emulated and real nodes in a same experiment. In case (a), a simulated node exchanges packets through a simulated link with an emulated node consisting of a real application and an emulated protocol stack. In case (b), a link-level emulated node exchanges packets through a real link provided by EMU with a real node.

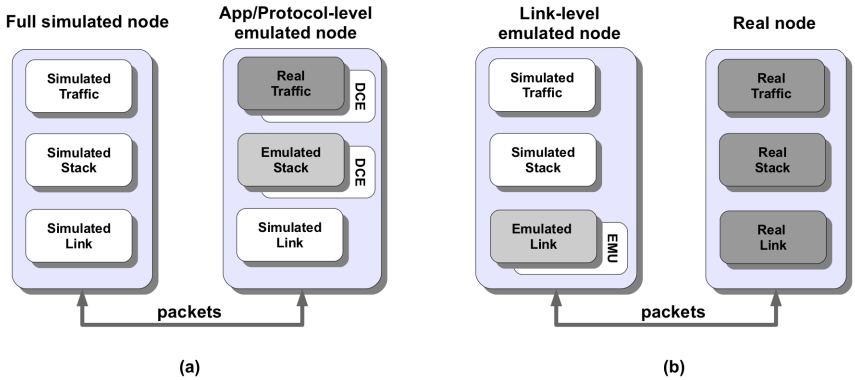


Figure 2: Ns-3 allows to exchange packets between simulated, emulated and real nodes: (a) a fully simulated node and an application/protocol-level emulated node using ns-3 Direct Code Execution (DCE), (b) a link-level emulated node using ns-3 Emulation (EMU) and a real node.

Components should be interoperable at different network layers, since different testbed block components work on different spaces (e.g., user-space and kernel-space). EMU is used for link-level emulations, to integrate a Network Interface Controller (NIC) and a ns-3 simulated protocol stack. Finally, in order to minimize user's effort to compose and control experimentation scenarios, a common script language is provided. Ns-3 and NEPI support the same scripting format to write experimentation scenarios, covering from simulation up to evaluation on real testbeds.

3.3. IDEV Use Cases

The IDEV framework can support a wide range of network research and development profiles, such as application developers, network protocol developers, and network designers working on topological build, network-service, and so on. In the rest of this section, we present three use cases to demonstrate the flexibility of our approach.

3.3.1. Application developer

Figure 3 shows the transitions along four different environments to support network application development, where the same application code can be evaluated across all environments. First, the application is executed in a highly controlled environment with simulated nodes, simulated protocol stacks, and simulated network links. Second, the application developer can use various emulation environments where only the network links and/or the channels are real. Third, the application can be evaluated in a fully real environment, using real nodes, real protocol stacks, and real network links. By playing with these alternatives it is possible to isolate diverse sources of problems and recreate failure scenarios.

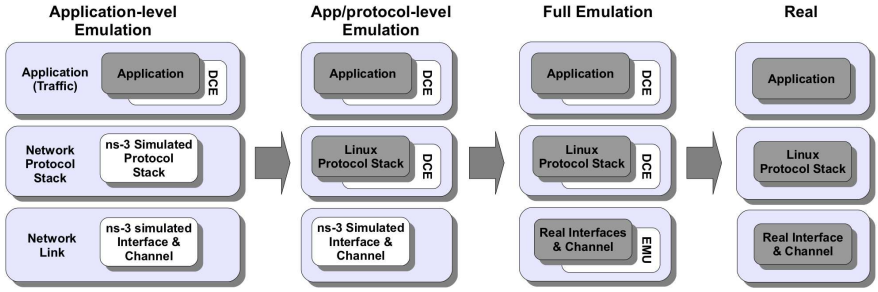


Figure 3: Possible IDEV scenarios for network application development.

Some representative examples of applications that fit the application development profile include network performance dependent applications, such as multimedia streaming, content sharing by peer-to-peer networks, and massive transactions (e.g., internet banking services and online games). In particular, when a multimedia streaming server or service is designed, the developer is usually interested in analyzing traffic characteristic, content popularity, data caching, and overall performance. Chesire et al. [13] presented

an empirical study using domestic streaming service to measure and analyze high data-rate multimedia streaming. However, they encounted difficulties to enhance performance and reduce requirements of the system. The reason is that the experimental environment in a laboratory is different than the actual deployment.

3.3.2. Network Protocol Developer

The IDEV framework can help developing network protocols at any layers including data-link and physical layers for both wireless and wired communications. As shown in Figure 4, researchers and developers start by constructing a simplified protocol module using simulation (e.g., ns-3), and then test it for proof concept in fully controllable environment. When the test fulfills the requirements in terms of performance and functionality on a simulation environment, this module is ready to be evaluated using real links provided by EMU.

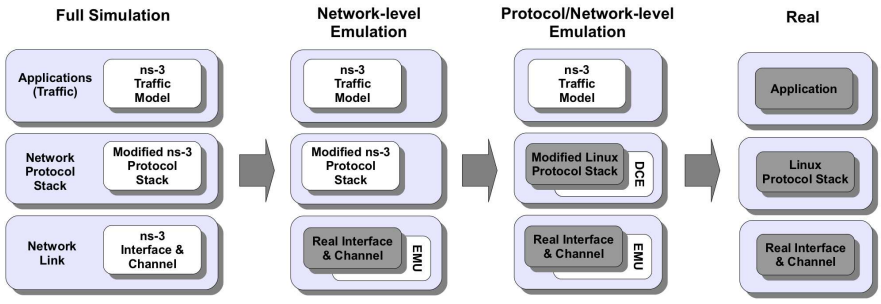


Figure 4: Possible IDEV scenarios for development of a new Layer 3 (L3) or Layer 4 (L4) protocol.

After these two steps, the protocol is ready to be re-implemented in kernel code. The developer can validate and debug the kernel modules using ns-3/DCE with traffic generators and virtual network channels. In the final stage, the customized Linux kernel can be evaluated with real applications and links on real machines, using DCE and EMU. To enable realistic evaluation of lower layers network protocols, we plan to equip the testbed nodes with Field Programmable Gate Arrays (FPGA) devices, such as Universal Software Radio Peripheral (USRP) [14], to develop network protocols on data-link and physical layers.

3.3.3. Network designer

The IDEV framework provides a complete planning solution for network designers. Network designers are usually responsible for estimating the required capacity of a new network infrastructure, drawing network topology maximizing cost-effectiveness, and testing interconnections between heterogeneous networks.

Figure 5 shows possible scenarios to evaluate a network protocol. Initially, the network designer can start with the simplest environment (i.e., fully simulated environment) to discover basic requirements. After refining the requirements, a real protocol stack can be added to increase the environment realism using DCE, followed by adding link-level emulation though EMU. Finally, the designer can replace all nodes with real nodes. If sufficient real nodes are not available, it is also possible to conceive a scenario where some of the nodes are real and other nodes are simulated to increase scalability.

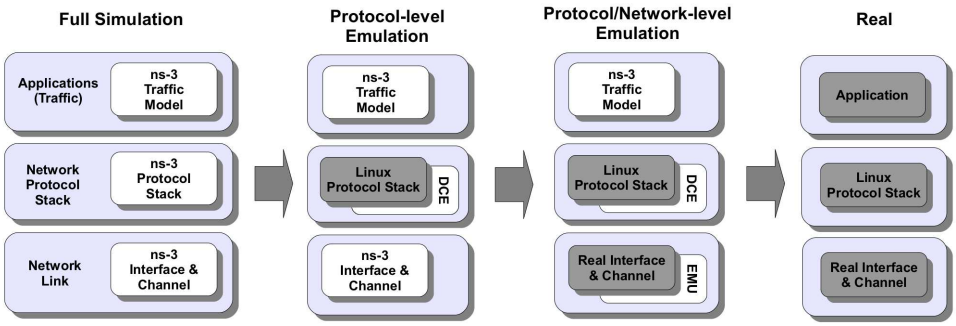


Figure 5: Possible IDEV scenarios for network design.

3.4. An Example of Iterative Development

The following example targets researchers working on transport and network layers. As shown in Figure 6, this example involves a series of experiments consisting of six environments from a pure ns-3 simulation to an experimentation in the wild through four different types of emulation.

The first test experiment shown in the left side of Figure 6, is a pure simulation using ns-3 components from the wireless channel up to the application layer. The second experiment consists of an application level emulation, where the ns-3 traffic generation model is replaced with a real application

configured with the same traffic parameters. The third experiment is a network level emulation using real Wi-Fi (IEEE 802.11) devices and channels, but the application and the network protocol stack are simulated. The fourth one, named Application and Link level emulation, uses simulation only for the network protocol stack. The fifth experiment, named full emulation, sets up the simulated protocol stack included in transport and network layers with real Linux kernels, but it still uses ns-3/DCE because of the flexibility to change Linux kernel modules. In the last experiment, all elements consist of real components from the channel to the application.

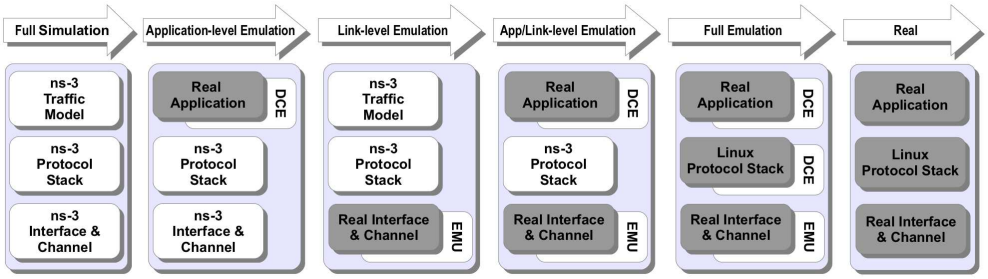
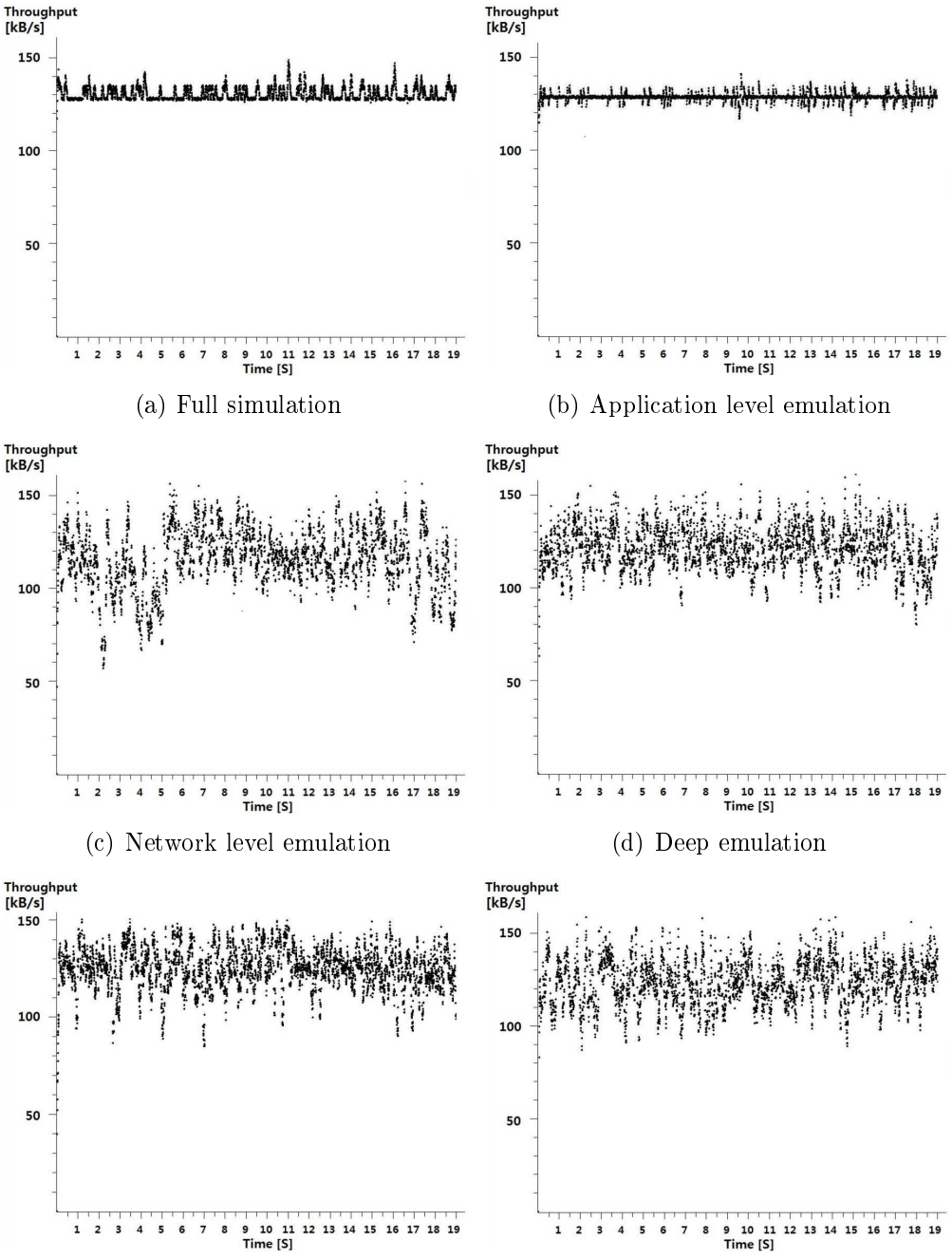


Figure 6: An example of IDEV for transport and network layers.

In this experimentation, the topology consists of a point-to-point connection between two IEEE 802.11 nodes, and they are connected to each other in the ad hoc mode. The distance between the two nodes is set to 5 meters in ns-3 for the first two experiments. In the first two experiments that use a simulated channel model, we use the log distance propagation model for describing the path loss, and set the Received Signal Strength Indicator (RSSI) at -60 dB to represent the wireless channel conditions. In the remaining four experiments, we place these two nodes at a distance that corresponds to the RSSI value used during the first two experiments.

We set the physical layer bit-rate using 802.11b at 2 Mbps for all experiments. One of the two nodes transmits a TCP stream, at a constant bit rate of 1 Mbps with a packet of size 1 kB for a duration 20 seconds. For such a traffic specification, we use a simple traffic generator provided by ns-3, for the full simulation and the application-level emulations. The remaining four experiments use a simple TCP application, generating the same traffic pattern used as for the first two experiments.



(a) Full simulation

(b) Application level emulation

(c) Network level emulation

(d) Deep emulation

(e) Full emulation

(f) Experiment in the wild

Figure 7: A comparison of throughput performance for different approaches. The experiments from (c) through (f) use a real wireless channel instead of a simulated model used at (a) and (b).

Figure 7 shows the throughput obtained on application level for all the experiments. The throughput patterns of the full simulation and of the application level emulation are very similar as shown in 7(a) and 7(b), respectively. In contrast, as seen in 7(c) through 7(f), the rest of the experiments exhibits significantly fluctuating throughput patterns, since the RSSI widely varies around ± 10, against the target value.

Indeed, the signal interference from outside should be eliminated to guarantee the trustiness of the experimentation. This kind of unpredictable and uncontrollable interference factor makes it difficult to draw consistent conclusions from the experiments. Therefore this example emphasizes the need to employ a controlled environment composed of Faraday cage and an anechoic chamber in our testbed, in order to enable wireless experiment repeatability. The shielded space can emulate the free-space channel model by insulating the external noise and absorbing the internal signal. However, such a controlled environment is not representative of real-life's wireless environments. In future, we plan to add controlled channel interferences in the shield space in order to increase realism of the wireless environments in the testbed, such as indoor space and urban street.

As shown in Figures 7(c) through 7(f), the experiments where the environments use real network devices do not exhibit a significant variation in the results. This means that the network protocol implementations from ns-3, the kernel (net-next-2.6) supported by ns-3/DCE [15], and the kernel (Fedora 10, kernel version 2.6.27) of the real node, behave similarly. Therefore, researchers have the possibility to compare the results obtained on these environments to those obtained when experimenting in the wild.

4. Testbed Architecture

In this section, we present a testbed architecture compliant with the IDEV approach. As shown in Figure 8, the testbed architecture is composed of three layers: User Interface, Back-end Infrastructure, and Testbed Resources. The user interacts with NEPI [16, 5], which provides the experiment description language and management framework used to conduct experiments. The back-end infrastructure layer operates as an intermediate layer between the user interface and the testbed resources. This layer is composed of testbed federation interface (SFA), a resource reservation tool (NITOS scheduler) and a resource control framework (OMF). The testbed resources layer comprises all physical resources.

The testbed architecture can also be divided into an experimental plane and a control plane. The former mainly carries experimentation control messages and input/output data (e.g., sample materials, measured results, and traces). The latter conveys resource information between the user interface and resource controller(s) of testbeds.

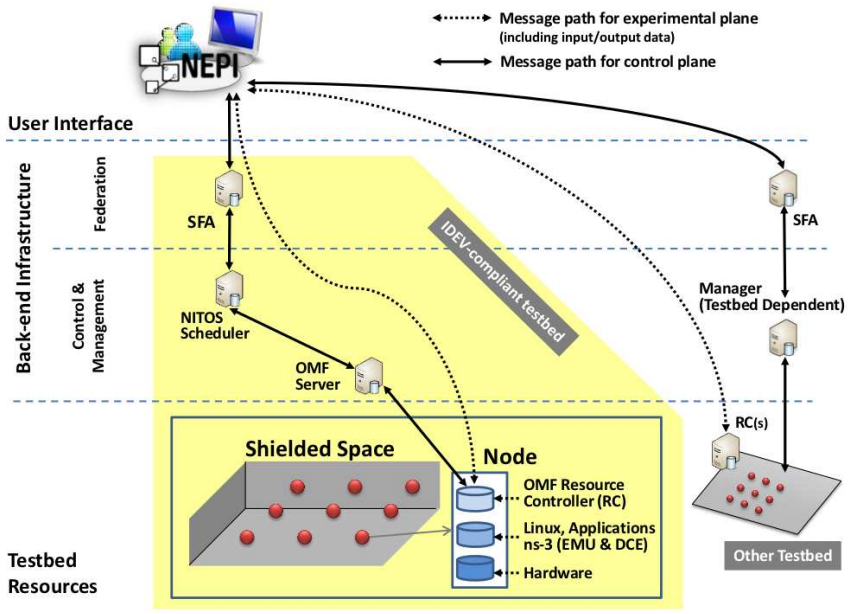


Figure 8: The testbed architecture is composed of three layers: User Interface, Back-end Infrastructure, and Testbed Resources.

The *experimental plane* is responsible for providing life-cycle management for experiments. This includes supporting all necessary functionalities to simplify the tasks of designing, deploying and controlling an experiment, and gathering the results generated during experiment execution. This plane is meant to abstract the researcher from the complexities of the underlying layers, alleviating the significant time requirements and human error associated with manual set up.

The *control plane* is responsible for restricting/granting access to the resources of the testbed (i.e., user authentication, communication with the different resources), and for identifying and allocating the resources required

for an experiment. It is also responsible for coordinating the federation with other testbeds, as it will be explained in more detail later in this section.

Finally, the physical infrastructure is composed of all physical components included in the testbed: the nodes, the NICs, the wireless channel, the Ethernet links, but also the infrastructure required to offer protection insulation from electromagnetic interference. The following subsections account for the details of the experimental plane, the control plane, and the testbed infrastructure.

4.1. Experimental Plane

The high complexity and variety of available network experimentation environments has lead the networking community to implement a multitude of tools to ease complex tasks such as experiment design, deployment, control and result collection. Among those tools, we chose the Network Experiment Programming Interface (NEPI) to manage experiments for our testbed because it was specifically conceived to interact with simulators, emulators and live testbeds, and to simplify the complex task of mixing those heterogeneous environments in a single experiment.

Experiments in NEPI are described as graphs of interconnected resources, where a resource can be any supported physical or virtual component (e.g., a PlanetLab node, a command line application or a ns-3 wireless channel). NEPI is able to control and interconnect these resources through dedicated entities called resource managers. The latter contains the necessary code to control the resources and communicate with them. For instance, in order to control a Linux host accessible through Secure Shell (SSH), NEPI provides a Linux host resource manager which is able to execute commands on the node using a SSH account. A specific NEPI entity, called the *Experiment Controller*, is responsible for interpreting the experiment graph and allocating, configuring and running the resources described by the user. In this way, the user can reuse experiment scripts, input data, and program source codes including applications and protocol stacks.

Currently, NEPI supports ns-3 resources for simulation, Nemu [17] resources for network emulation, PlanetLab [9] and OMF [4] resources for live experimentation. Supporting OMF resources was straightforward by re-using an existing Python XMPP [18] client and implementing resource managers to extend NEPI's API. To benefit from ns-3 resources, we took advantage of the ns-3 Python bindings and implemented resource managers able to create ns-3 objects, interconnect them and configure them through some bindings. On

the other hand, we faced difficulties in adding the support for DCE. Indeed, as Python bindings for DCE are not yet available, interacting with C++ ns-3 libraries from Python code is not trivial. Supporting DCE in NEPI is still an ongoing effort, however the integration challenges we face are only technical.

4.2. Control Plane

Designing a control plane for a network testbed, involves the definition of a control framework to manage user accounts and physical resources, takes into account the specificities of the facility. In our case, one strong requirement for the control plane is the ability to manage wireless resources, as well as to expose a generic and easy to use interface for the user, in order to foster an open facility reach for a wider users community.

Among available approaches, OMF is the control and management framework that best suits our testbed needs. OMF is a testbed management software which supports the management and the automatic execution of experiments on a wireless networking testbed. An OMF testbed is made of a number of nodes, equipped with wireless interfaces available for running the users experiments, and core software components that provides the infrastructural services needed to control the system and its configuration. These software components support all the phases of an experiment life-cycle, from the resources provisioning to the collection of experimental data.

OMF includes an Experiment Controller (EC), representing also the interface to the user, which is fed with an user-provided experiment description and takes care of orchestrating the testbed resources to run accomplish the required experimentation scenario. The EC interacts with the Aggregate Manager (AM), the entity responsible of managing and controlling the status of the testbed resources. The EC also interacts with Resource Controllers (RCs), which are components running within each of the testbed nodes and responsible for performing local configuration steps.

Communication between the EC and RCs is done via a Publish-and-Subscribe (Pub-Sub) extension of the eXtensible Messaging and Presence Protocol (XMPP) [18]. All entities in OMF register with the XMPP server and subscribe to a set of PubSub topics.

OMF is currently used in many wireless testbeds (e.g., NITOS [19], ORBIT [20], and NICTA [21]), and has a large group of active users and contributors. Among other things, it enables to deploy user-modified operating system images in the nodes to easily configure resources, to monitor experiments and collect measurements.

On the other hand, the users community has strongly expressed the need to enable cross-testbed experiments [22], allowing to combine heterogeneous resources coming from different testbeds into a single experiment. This allows not only to increase the size of the user community of each testbed, but also to run larger scale experiments with more resources than what a single testbed could offer, and with wider variety of technologies.

The idea of testbed federation has emerged offering a seamless interconnection between heterogeneous testbeds, providing a common framework to provision the resources, and opening the testbed to the external world.

For the above reasons, we decided to use the Generic SFA Wrapper (aka SFAWrap), in order to expose our testbed to the global federation, and to provide interfaces that are compliant with the Slice-based Federation Architecture (SFA). SFA has been specified to provide a minimal set of functionalities, a "thin waist", that a testbed needs to implement in order to enter into a global federation, while the Generic SFA Wrapper is one of the most visible and renowned reference implementation. It is successfully used to federate a variety of testbeds, including Planetlab [9], NITOS [19], Senslab [23], GENI [24], G-Lab [25], GpENI [26], and Federica [27, 28].

The Generic SFA Wrapper comprises of three building blocks, namely: Registry (R), Aggregate Manager (AM), and Slice Manager (SM). The Registry implements the *Registry API* and is responsible of maintaining and serving SFA records (Authorities, Users and Slices) and of issuing the related certificates and credentials. The Aggregate Manager (AM) implements the *GENI AM API* [29] and is responsible of performing all the slice instantiations. Also, it allows testbeds to advertise their resources and attach these latter to slices. Finally, the Slice Manager (SM), which also implements the *GENI AM API*, acts as a proxy aware of a pre-configured set of other services, such as Aggregate Manager or Slice Manager [30].

In order to enable OMF and SFA to work together and provision testbed resources, we need to add the functionnalities to identify available nodes and reserve them for a user and a specific time slot. OMF considers resources as exclusive, preventing possible problems arising from concurrent users using the same wireless network cards or channels, which could affect the expected behavior of experiments. This means that OMF resources have to be reserved before access to the resources is granted by SFA. So, a scheduler is required to take care of identifying and reserving testbed resources for a particular user-provided time slot.

As no scheduler is included in OMF, we decided to use NITOS, a sched-

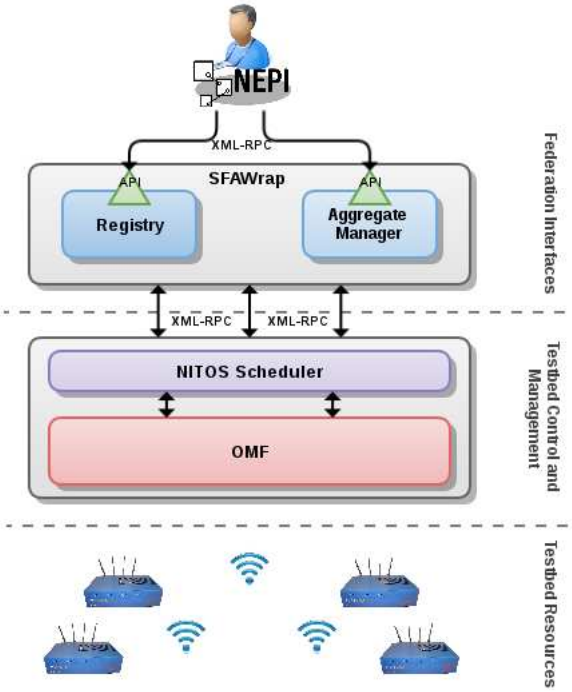


Figure 9: Overall architecture of our testbed control plane.

uler compatible with OMF that enforces resource reservation, supports the concept of *Slice*[1] and implements a spectrum slicing scheme preventing experiments to interfere with each others. Other available OMF-based schedulers, like ORBIT or NICTA, do not support the concept of slice, and/or do not enforce resources reservation but reserve them in a "gentlemen's agreement" fashion.

The NITOS scheduler performs resource discovery, reservation, and controls the access to the resources. It includes a front-end for reserving resources for limited time intervals, a back-end to handle access to the resources and a mechanism to monitor and control the resources access policies in order to prevent possible conflicts or malicious actions.

Furthermore, it implements a spectrum slicing scheme that enables a bet-

[1]A Slice is a group of resources allocated to a particular experiment and distributed across several nodes.

ter usage of wireless testbeds, by allowing to run several wireless experiments at the same time without interfering with each other [31].

Overall, integrating SFAWrap with the NITOS Scheduler and OMF was challenging. Apart from the technical engineering aspects that bundles defining the communication interfaces and mechanisms, we had to address the specificities of our testbed. In particular, we extended the resource description scheme of SFAWrap to support wireless nodes and wireless channels. We also introduced time-based reservation of the resources into SFAWrap to book resources for specific time slots. Figure 9 illustrates the control plane of our testbed.

4.3. Testbed Infrastructure

In the quest of reproducible research, it is important to note that the infrastructure in which the experiment is deployed has an important impact on the quality and the coherency of the results obtained. Compared to simulation, where the environment is highly controllable, a real infrastructure relies on its environment and consequently, requires more precaution.

Indeed, experimentation results obtained with real conditions are highly dependent on the network environment characteristics, and unexpected changes in the environment conditions can impact the outcome of the experiment in a random way. Wireless networks in particular are challenging because the communication channel is naturally exposed to uncontrolled electromagnetic interferences. To limit the impact of these uncontrolled interferences as well as get to closer to the reproducible experiments requirements, some insulation's technologies are necessary to insulate the wireless channel.

Electromagnetic insulation of the testbed can be obtained through two types of solutions: insulation from outside and insulation from both inside and outside. The first solution usually consists of a Faraday cage providing insulation from external electromagnetic interference only, whereas the second one consists of an anechoic chamber providing insulation from external electromagnetic waves as well as internal reflected ones, by using absorbers placed on the walls. Anechoic chambers are categorized in different groups [32] according to which sides of the chamber contain absorbents. As our testbed aims to provide reproducible network experiments, we need a Fully Anechoic Chamber (FAC) which covers the four walls, the floor, and the ceiling.

Many different materials exist to build an anechoic chamber, each one with different performance and cost. Usually, both Faraday cage and ane-

choic chamber, use a thin layer of copper. A copper layer of 0.3 mm covering all the surfaces can reduce the interference up to 100 dB in the WiFi frequency band. Concerning the absorbent walls, foam pyramid absorbers are usually utilized. The length of the foam pyramids determine the absorption of the reflected wave for a frequency range. An absorber size of 200mm is convenient to perform efficiently in WiFi environments.

By building the testbed inside a fully anechoic chamber with the characteristics described above, we expect to provide adequate conditions for achieving, as much as possible, reproducible WiFi experiments.

The choice of the wireless nodes in the testbed needs to be carefully made. First of all, they must be able to handle experiments requiring possibly a high CPU utilization, and running for long periods of time. Moreover, remote control of devices are required to enable remote hard reboot of the nodes in case of failures. Finally, each node needs at least three Ethernet and one wireless network interfaces: one for managing the nodes , one for the remote control, and is (wired and wireless) dedicated to the experiments.

We evaluated different nodes according to the criteria described above and compared them with the technologies that are already deployed in well-known testbeds (like iMinds [33] or NITOS [19]). We finally decided to use about 40 Icarus NITOS nodes [34] deployed in a grid topology. This type of node uses Intel Core i7 at 3.40 GHz, granting enough power to support experiments requiring high CPU utilization. In contrast, ORBIT nodes [35], which are mainly deployed in wireless testbeds, are not powerful enough for our needs (with their 1 GHz processor).

We are also investigating some others components to install in the testbed, such as "noise generator" to enable the possibility to add controlled interference environment depending on the experiment scenario. In the future, we will also investigate how to emulate mobility in the testbed, e.g., using an approach similar to iMinds [33, 36].

5. Conclusion

In this paper, we propose the Iterative Development of Network Protocols (IDEV) framework that allows combining, in a unique test environment, fine-grained simulation, application-level emulation, protocol-level emulation, link-level emulation, and experimentation in the wild. This addresses the need for more flexible network evaluation environments, where

researchers can create custom network evaluation and design their experiments with any level of realism.

We present IDEV through use cases and detail the different components that take part in its architecture. Then, we describe in detail an IDEV testbed which supports automate all network experiment steps, and allows federation with other testbeds. We believe the network community can benefit from this framework to design more easily new network protocols and applications, as well as to generate reproducible performance results, which is very challenging to achieve in wireless environments.

Acknowledgement

The authors would like to thank the anonymous reviewers and the editors for their valuable comments and suggestions to improve the quality of the paper. They are also grateful to Dr. Ashwin Rao for useful comments and proofreading.

References

[1] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, A. Joglekar, An Integrated Experimental Environment for Distributed Systems and Networks, ACM SIGOPS Operating Systems Review 36 (2002) pp. 255–270.

[2] The ns-3 Network Simulator, URL "`http://www.nsnam.org/`", last visited on 20/12/2013.

[3] H. Tazaki, F. Urbani, E. Mancini, M. Lacage, D. Camara, T. Turletti, W. Dabbous, Direct Code Execution: Revisiting Library OS Architecture for Reproducible Network Experiments, in: Proceedings of the 9th International Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2013, pp. 217–228.

[4] T. Rakotoarivelo, M. Ott, G. Jourjon, I. Seskar, OMF: A Control and Management Framework for Networking testbeds, ACM SIGOPS Operating Systems Review (2009) pp. 54–59.

[5] C. Freire, A. Quereilhac, T. Turletti, W. Dabbous, Automated Deployment and Customization of Routing Overlays on PlanetLab, in: Proceedings of the 8th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM), 2012, pp. 240–255.

[6] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, N. McKeown, Reproducible Network Experiments using Container-based Emulation, in: Proceedings of the 8th international conference on Emerging networking experiments and technologies (CoNEXT), 2012, pp. 253–264.

[7] R. Ricci, J. Duerig, P. Sanaga, D. Gebhardt, M. Hibler, K. Atkinson, J. Zhang, S. K. Kasera, J. Lepreau, The Flexlab Approach to Realistic Evaluation of Networked Systems, in: Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2007.

[8] The ns-3 EMU Module, URL "`http://www.nsnam.org/docs/release/3.11/models/html`
last visited on 20/12/2013.

[9] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, M. Bowman, PlanetLab: An Overlay Testbed for Broad-coverage Services, SIGCOMM Computer Communication Review 33 (2003) pp. 3–12.

[10] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, et al., Advances in Network Simulation, Computer 33 (5) (2000) pp. 59–67.

[11] A. Bavier, N. Feamster, M. Huang, L. Peterson, J. Rexford, In VINI Veritas: Realistic and Controlled Network Experimentation, ACM SIGCOMM Computer Communication Review 36 (2006) pp. 3–14.

[12] Pcap and Tcpdump, URL "http://www.tcpdump.org/", last visited on 20/12/2013.

[13] M. Chesire, A. Wolman, G. M. Voelker, H. M. Levy, Measurement and Analysis of a Streaming-media Workload, in: Proceedings of the 3rd Conference on USENIX Symposium on Internet Technologies and Systems (USITS), 2001, pp. 1–12.

[14] USRP Device, URL "http://gnuradio.org/", last visited on 20/12/2013.

[15] Ns-3/DCE, URL "http://www.nsnam.org/overview/projects/direct-code-execution last visited on 20/12/2013.

[16] A. Quereilhac, C. Freire, M. Lacage, T. Turletti, W. Dabbous, NEPI: An Integration Framework for Network Experimentation, in: Proceedings of the 19th International Conference on Software, Telecommunications and Computer Networks (SoftCom), 2011, pp. 1–5.

[17] M. Ferrari, NEPI: An Experiment Plane for Experimentation Testbeds, INRIA Technical Teport.
URL "http://hal.inria.fr/inria-00601848"

[18] XEP-0060: Publish-Subscribe, URL "http://xmpp.org/extensions/xep-0060.html", last visited on 20/12/2013.

[19] Network Implementation Testbed Laboratory, URL "http://nitlab.inf.uth.gr/NITlab/", last visited on 20/12/2013.

[20] ORBIT Website, URL "http://www.orbit-lab.org/", last visited on 20/12/2013.

[21] NICTA Website, URL "http://www.nicta.com.au/research/projects/tempo", last visited on 20/12/2013.

[22] R. Ricci, J. Duerig, L. Stoller, G. Wong, S. Chikkulapelly, W. Seok, Designing a Federated Testbed as a Distributed System, in: Proceedings of the 8th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM), 2012, pp. 321–337.

[23] SensLab Website, URL "http://www.senslab.info/", last visited on 20/12/2013.

[24] M. Berman, J. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, I. Seskar, GENI: A Federated Testbed for Innovative Network Experiments, Computer Networks (2014) –.

[25] D. Schwerdel, B. Reuther, P. Müller, T. Zinner, P. Tran-Gia, Future Internet Research and Experimentation: The G-Lab Approach, Computer Networks (2014) –.

[26] D. Medhi, B. Ramamurthy, C. Scoglio, J. P. Rohrer, E. K. Çetinkaya, R. Cherukuri, X. Liu, P. Angu, A. Bavier, C. Buffington, J. P. Sterbenz, The GpENI Testbed: Network Infrastructure, Implementation Experience, and Experimentation, Computer Networks (2014) –.

[27] Federica website, URL "http://www.fp7-federica.eu/", last visited on 20/12/2013.

[28] M. Campanella, F. Farina, The FEDERICA Infrastructure and Experience, Computer Networks (2014) –.

[29] GENI Aggregate Manager API, URL "http://groups.geni.net/geni/wiki/GAPI_AM_API", last visited on 20/12/2013.

[30] L. Peterson, S. Sevinc, S. Baker, T. Mack, R. Moran, F. Ahmed, PlanetLab Implementation of The Slice-Based Facility Architecture, URL "http://www.protogeni.net/wiki/AMAPI" (Jun. 2009).

[31] A.-C. Anadiotis, A. Apostolaras, D. Syrivelis, T. Korakis, L. Tassiulas, L. Rodriguez, M. Ott, A New Slicing Scheme for Efficient Use of Wireless Testbeds, in: Proceedings of the 4th ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH), 2009, pp. 83–84.

[32] K. Liu, Anechoic Chambers for Evaluating and Validating Telecommunication Equipment, Safety and EMC MagazineLast visited on 20/12/2013.

[33] iMinds, URL "http://www.crew-project.eu/wilabt", last visited on 20/12/2013.

[34] Icarus Node Description, URL "http://nitlab.inf.uth.gr/NITlab/index.php/hardwar last visited on 20/12/2013.

[35] ORBIT Node, URL "http://groups.geni.net/geni/wiki/GeniAggregate/OrbitWirele last visited on 20/12/2013.

[36] S. Bouckaert, B. Jooris, P. Becue, I. Moerman, P. Demeester, The IBBT w-iLab.t: A Large-scale Generic Experimentation Facility for Heterogeneous Wireless Networks, in: Proceedings of the 8th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM), 2012, pp. 7–8.