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 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 to improve 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 to refine and improve 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 are fairly accurate. Previous approaches to tackle this problem such as Emulab [1] allow to provide support for simulation and emulation in a same facility, making the transition from one to the other relatively easy by using the same script language for both emulation and simulation scenarios. However, neither Emulab, nor any other solution that we are aware of, allows to provide fine grain control on the realism level of the evaluation environment.

This is to say, in a way that empowers the researcher to choose gradually which parts of the experiment (application, transport and network protocol, link and physical layers) should be simulated or real, with the possibility to go easily from full simulation (all parts simulated) to live experimentation on real nodes (all parts with real code and hardware). The goal of this paper is to propose a network experimentation methodology (tools and testbeds) that answers this problem.

This methodology call IDEV for Iterative Development of Network Protocols, addresses therefore the need for more flexible experimentation environments, where researchers can design their experiments with any level of realism, allowing them to transit from a purely simulated to a purely realistic environment in a very smooth way. To illustrate the interest of this approach let's take the following example scenario: performance evaluation of video streaming over MultiPath TCP (MPTCP) [2] in wired and wireless multipath network. This could be done with a full simulation, by implementing the video streaming application code and the MPTCP protocol in a network simulator such as ns-3 and using the simulator's network, link, physical layers and channel models. We could also use the real code of streaming applications and/or the MPTCP Linux code by integrating them in ns-3 (we explain later how this can be done easily). The next step could be also to run ns-3 in real time mode on top of a real Wi-Fi and Ethernet networks. At each step or *iteration*, it is possible to use either the simulated or the real code part of the protocol stack. This methodology aims to account for the need of reproducible research to ensure scientific rigor.

In this work, we have chosen to focus on the wireless aspect of network evaluation, since we believe that it is sufficiently complex to present the biggest challenge in terms 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 are 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.

In order to materialize our proposed methodology, we present a set of tools (a framework) and a testbed architecture that organized together in the right way to support the IDEV concept. The proposed framework and testbed leverage on existing technologies, such as the ns-3 network simulator, and Direct Code Execution (DCE) [3] for simulation and emulation support, as well as on the OMF (cOntrol and Management Framework) [4] to support experimentation on real hardware. The testbed itself, includes about 40 high performance wireless nodes and an RF anechoic chamber built into a Faraday cage to enable reproducible wireless experimentation results.

The proposed architecture also makes use of the Network Programming Interface (NEPI), to simplify the description, execution and control of the same experiment scenario over different environments. NEPI provides a high-

level API to interact with resources from different evaluation environments, notably the ns-3 simulator and OMF (cOntrol and Management Framework). Through this API, it automates network experiment steps, such as experiment set-up, application installation, error detection, and collection of results.

The rest of the paper is organized as follows. In section 2, we provide an overview of a few representative network experimentation solutions and discuss differences with our proposed methodology and framework. 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. Finally, in section 5 we draw conclusions.

2. Related work

In the network research community, reproducibility is becoming more popular, and some emulation based tools are available to reproduce experimental results. Emulation is the combined use of one or more simulated parts and one or more real code and/or hardware parts in the same experiment. Recently, Nikhil Handigol et al. [5] in 2012, replayed about a dozen of network experiments using Mininet, a container-based network emulator. Mininet enables to run standard Unix/Linux network applications and real Linux kernels, in a lightweight and a cheap way. However, Mininet works "only" with emulated low layers, for example it cannot use real devices and links. Also, this tool does not allow to easily transform 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 [6], Emulab [1], Flexlab [7].

Ns-3 supports not only simulation, but also elaborate emulation environments 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 easy to reuse the same script in the Emulab testbed and the ns-2 simulator. However, Emulab is not able to provide realistic and reliable network conditions. In the Emulab testbed, wired nodes are connected through Ethernet links, but the channel condition 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 exposed in an open space, surrounded by 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 no more 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 real testbed (e.g., PlanetLab) are time-varying, 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.

Overall, none of these approaches provides support for easy gradual increase of the experiment realism level, and this is what IDEV aims to offer.

3. Iterative Development of Network Protocols (IDEV)

Network simulation is a technical process, in which the behavior and outcome of a target network is predicted by calculating interactions between the different network entities, such as host and protocol stack. In general, network simulators 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 also provide a simplified view of the rules that govern component interaction, and so the realism that they can achieved is still limited.

Network emulation combines realism and controllable modelization, using simulation and real software and/or hardware, but it does it in limited and specific ways. Emulation is often used for performance evaluation, such as for predicting the impact of changes in existing protocols and application or optimizing 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

pre-captured traffic. 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 (i.e., applications, protocol stacks, and network interfaces), but also environmental conditions identically 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 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 way, 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 sections we describe the IDEV methodology through use cases and explain the different components that take part in 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 applications, network protocol stack, and network infrastructure can be independently specified by the user, to be simulated, emulated or 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.

3.1.1. Simulation components

Simulated components leverage on the ns-3 network simulator [6], a discrete event network simulator that provides fairly detailed models for wireless devices and channels. The ability of ns-3 to generate simulated traffic, where packets match real network packets, can be exploited to integrate ns-3 simulations with real networks, at the network device level. These features make ns-3 a very versatile test environment, and thus a key component for our framework.

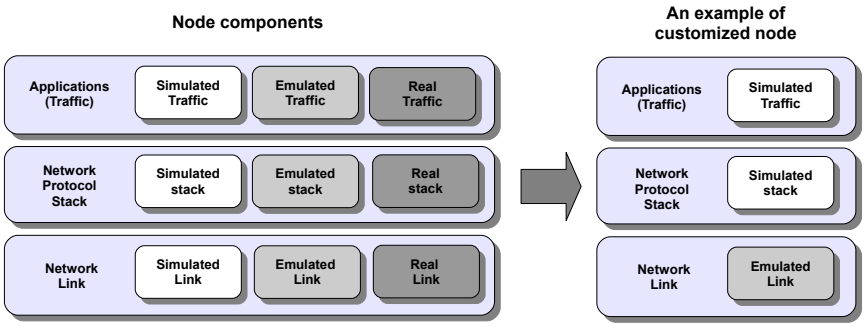


Figure 1: simulated, emulated, and real constructive blocks are used to form custom test environments.

3.1.2. Emulation components

To provide emulation components, we take advantage of a recent extension to the ns-3 simulator, the DCE [3], which allows to run unmodified C++ applications and protocols inside a simulation. DCE implements sophisticated mechanisms to interleave execution time slots of application code and simulation events. It provides a special implementation of the C standard library, and other Linux-specific libraries, that override specific system calls with ns-3 compatible operations. In this way invocations to system calls, such as UNIX *gettimeofday()*, are then translated into ns-3 calls, which return the simulation time instead of the system wall-clock time.

DCE enables the use of both user-space and kernel-space protocol implementations inside a ns-3 simulated node. It implements complex techniques to isolate global and static variables on multiple instances of a same protocol implementation. In this manner, different nodes in the same ns-3 simulation process can 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] completes the emulation capabilities of the framework, by allowing to transparently attach a ns-3 simulated node to a physical wireless network interface, and thus 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. Internally 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 subtract the traffic portion addressed to its own IP address, and inject it in 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 one will ignore this traffic, making possible the coexistence of simulated and real networks.

3.1.3. Real components

The IDEV testbed provides physical nodes to be used as realistic components. All nodes run a Linux operating system, making it possible to experiment with any Linux compatible application in the testbed. Furthermore, the testbed allows to deploy user modified Linux images on the nodes, allowing to experiment with modified Linux network protocol stacks. All nodes will be 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 be a general solution for building custom network evaluation environments that mix simulation, emulation and realistic components. For this, it must provide seamless integration between components of those different types by supporting: 1) realtime simulation capabilities and 2) an integrated packet representation.

Realtime simulation capabilities are necessary in order to synchronize simulated components with real hardware, such as real link and traffic. The ns-3 simulator [6] not only supports realtime simulation, but also generates real packets that can be seamlessly injected into a real network device or read by tcpdump. Ns-3 supports the pcap standard format [12] to capture and store packets in a compatible way with real network devices and protocol stacks. Through EMU [8] and DCE [3], ns-3 enables integration with real network-links, applications and protocol stacks.

Figure 2 illustrates two cases where synchronized time and packet representation are necessary to integrate simulated, emulated and real nodes in a same experiment. In case (a), a fully simulated node exchanges packets with an emulated node consisting of a real application and an emulated protocol stack, through a simulated link between them. In case (b), a link-level emulated node exchanges packets with a real node, through a real link provided by EMU [8].

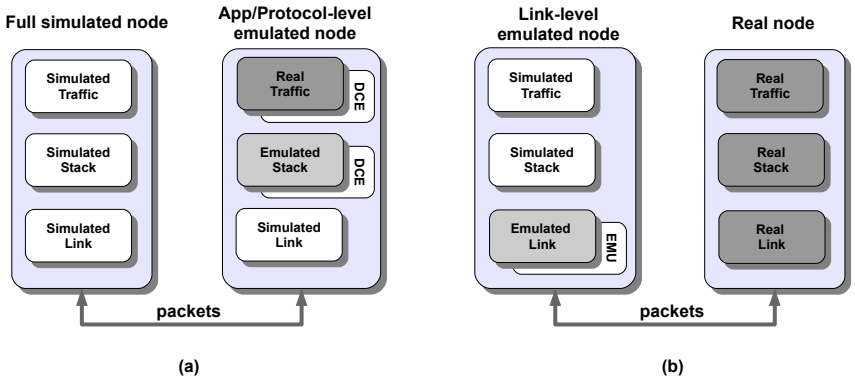


Figure 2: Ns-3 allows to exchange packets between simulated, emulated and real nodes: (a) a fully simulated node and an application/protocol-level emulated node, (b) a link-level emulated node and a real node.

Components should be interoperable at different network layers, since different testbed block components work on different spaces (i.e., user-space and kernel-space). To integrate a real application with a emulated protocol stack, we take advantage of DCE, which allows to run user applications written in C/C++ on top of a ns-3 simulated protocol stack without any modification. In addition, DCE enables to use network protocol stacks implemented on either user space or kernel space, and it protects global and static instances created by the same protocol stacks. In this way, DCE guarantees that multiple nodes involved in a same ns-3 experiment can use independent network protocol instances without interfering each other.

EMU is used for link-level emulations, to integrate a Network Interface Controller (NIC) and ns-3 simulated protocol stack. It can replace a simulated physical layer (or up to MAC sub-layer for most commercial products) and approximated channel models.

Finally, in order to minimize user's effort to compose and manage the experimentation scenarios, a common script language should be provided. Ns-3 supports the same scripting format to write experimentation scripts, covering from simulation up to evaluation on real testbeds.

3.3. IDEV Use Cases

The IDEV framework supports a wide range of network research and development profiles, such as application developer, protocol developer, and

network designer. In the rest of this section, we describe 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. Initially, the application is executed in a highly controlled environment with simulated nodes, simulated protocol stacks, and simulated network links. Then, the application developer can use various emulation environments where only the network links and/or the channels are real. Finally, 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.

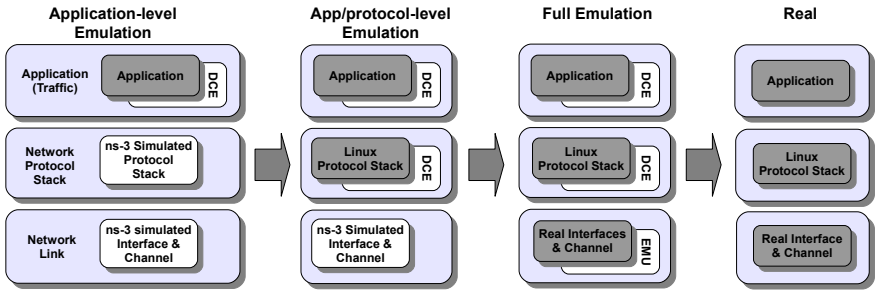


Figure 3: Possible IDEV scenarios for network application development.

Some representative examples of applications that fit the application development profile are: 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 analysing traffic characteristic, content popularity, data caching, and overall performance. Maureen Chesire et al. [13] presented an empirical study using domestic streaming service to measure and analyze massive multimedia streaming. However, they encounted difficulties to modify the system for enhancing performances or for reducing requirements. The reason is that the experimental environment in a laboratory can be far from 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. Initially, 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.

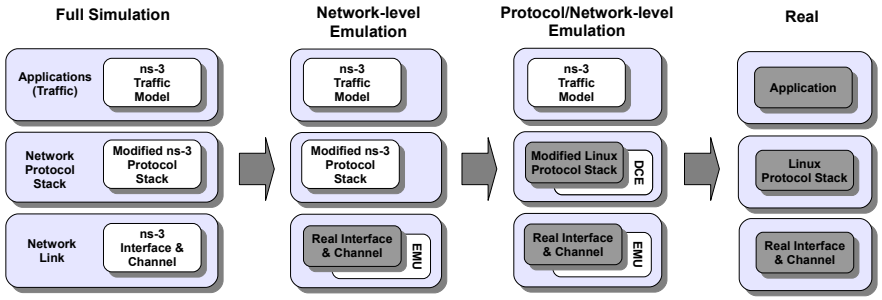


Figure 4: Possible IDEV scenarios for development of a new L3 or L4 protocol.

After these two steps, the protocol is ready to be re-implemented in kernel code. The developer can validate and debug the kernel modules within a ns-3/DCE emulation using traffic generators and virtual network channels. In the final stage, the customized Linux kernel, containing the first development version, 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 also 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 environment transitions to evaluate a network design. 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 by real nodes. If not enough real nodes are available, it is also possible to conceive a scenario where some of the nodes are real and other nodes are simulated to achieve scalability.

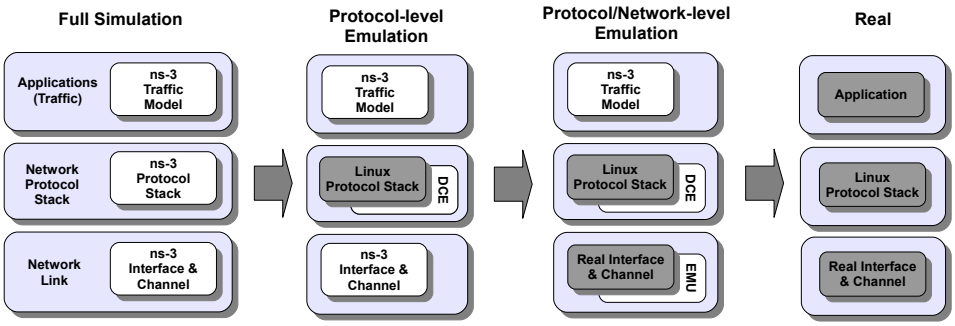


Figure 5: Possible IDEV scenarios for network design.

3.4. An Example of Iterative Development

We show an example of the proposed framework for researchers working on transport and network layers. As shown in Figure 6, this example involves a series of experiments, which consists of six environments from a pure ns-3 simulation to a full real environment through four different types of emulations.

The first test environment 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 environment is an application level emulation, where the ns-3 traffic generation model is replaced with a real application configured by the same traffic parameters. The third environment shows a network level emulation using real Wi-Fi (IEEE 802.11) devices and channels, but application and network protocol stack are simulated. The fourth one, deep emulation, uses simulation only for the network protocol stack. The full emulation set up the simulated protocol stack included in transport

and network layers with real Linux kernels, but still uses ns-3/DCE because of the flexibility to change Linux kernel modules. In the last environment, all elements consist of real components from the channel to the application.

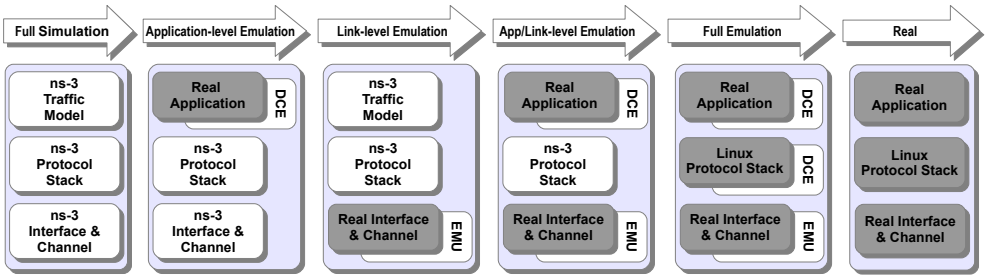


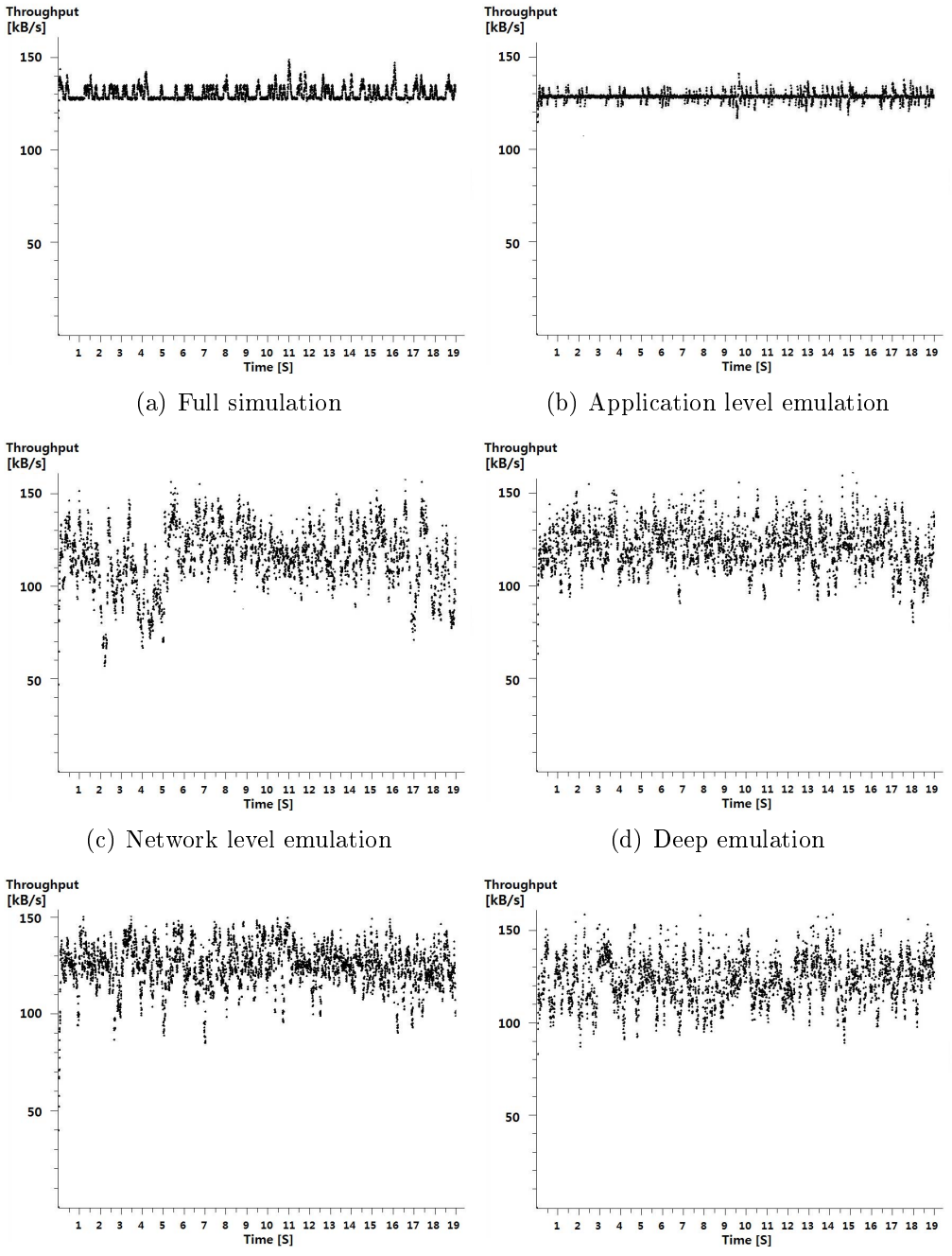
Figure 6: An example of IDEV for transport and network layers.

The topology consists of a peer-to-peer connection between two IEEE 802.11 ad hoc nodes without mobility. The distance between the two nodes is 5 meters in the first two experiments. We use the log distance propagation model, for describing the path loss in the first two experiments using a simulated channel model.

We set Received Signal Strength Indicator (RSSI) at -60 dB, to represent the wireless channel conditions in the first two environments using an ns-3 simulated channel. In the remaining environments, real wireless device are used. The two nodes are approximately in distance away corresponding to the same RSSI degradation used in the simulated channel environments.

The physical layer bit-rate using 802.11b is set to 2 Mbps for all experiments. One of those nodes transmits a TCP stream, which has been set to a constant bit-rate (1 Mbps) and a constant packet size (1 KB) for 20 seconds. With this traffic specification, we use a simple traffic generator provided by ns-3, for the full simulation and the application-level emulation. The rest of experiments use a simple TCP application, generating the same traffic pattern used that for the simulation environments.

Figure 7 shows user data throughput obtained for the experiments in each environment. 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 patterns. The observed RSSI widely varies around ± 10, against the desired level.



(a) Full simulation

(b) Application level emulation

(c) Network level emulation

(d) Deep emulation

(e) Full emulation

(f) Real experiment

Figure 7: A comparison of throughput performances for different approaches

As the RSSI variation shows, 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 a consistent conclusion from outcomes produced by iterating experiments. Therefore this example emphasizes the need to employ a Faraday cage and an anechoic chamber in our testbed, in order to enable wireless experiment repeatability. However, the shielded space by the cage and the chamber can emulate the free-space channel model by insulating the external noise and absorbing the internal signal, on the contrary it cannot represent real-life's wireless environments. In future, we plan to develop a scheme controlling channel conditions and describing realistic wireless environments in the shielded space.

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 fairly compare the results obtained on these environments to those generated in real networks.

4. Testbed Architecture

In this section, we describe in detail 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, 17], which provides the experiment description language and management framework to conduct experiments. The back-end infrastructure layer functions as an intermediate layer between the user interface and the testbed resources. This layer is composed by a testbed federation interface (SFA), resource reservation tool (NITOS scheduler) and resource control framework (OMF). The testbed resources layer comprises all physical resources.

The architecture can also be seen as divided into experimental plane and 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.

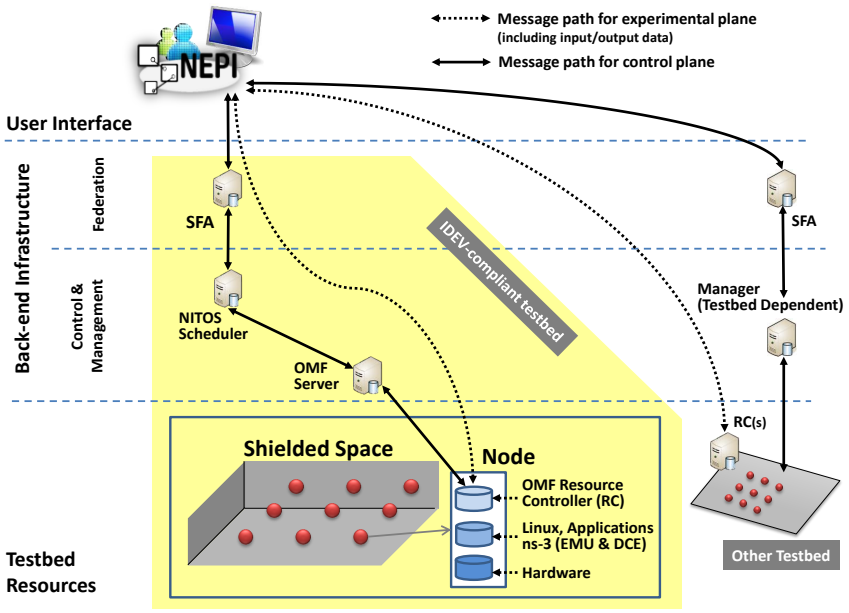


Figure 8: The testbed architecture is composed of three layers: user interface, back-end infrastructure, and testbed resources.

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 lengthy time requirements and error incidence associated with manual operation.

The control plane is responsible for restricting/granting access to the testbed physical resources (i.e., user authentication, communication with the resources), and for listing 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 by all physical components that belong to the testbed. These include the nodes, the NICs, the wireless channel, Ethernet links, but also the infrastructure required to achieve insulation from electromagnetic interference. The requirement for electro-

magnetic isolation responds to the need to enable wireless experiment repeatability in a realistic uncontrolled wireless environment. On this purpose, we plan to build the testbed inside an anechoic chamber. 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) [16, 17] to manage experiments for our testbed.

NEPI 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. This ability to handle heterogeneous experimentation environments is what makes NEPI an adequate candidate to support the proposed IDEV framework.

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, a ns-3 wireless channel, etc). NEPI is able to control and interconnect these resources through dedicated entities called resource managers. They contain the necessary code to control and communicate with the resources. For instance, in order to control a Linux host accessible through an SSH account, NEPI provides a Linux host resource manager which is able to execute commands on the node using the SSH account. A special NEPI entity, the *Experiment Controller*, is responsible for interpreting the experiment graph and allocating, configuring and starting the resources described by the user. By those functions and abilities, 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 [18] resources for network emulation, PlanetLab [9] and OMF [4] resources for live experimentation. Supporting OMF resources was straightforward to obtain, we did it by re-using an existing Python XMPP [19] client and implementing resource managers to extend NEPI's API. To support 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 using the

bindings. On the other hand, we faced difficulties in adding support for DCE. Indeed, Python bindings for DCE are not available and interacting with C++ ns-3 libraries from Python code is not trivial. Furthermore, interconnecting DCE simulated components with real hardware is not a trivial task and requires important engineering work. 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 networking testbed, involves the definition of a control framework to manage user account and physical resources, that 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 exposing a generic and easy to use interface for the user, in order to foster an open facility that would reach for a wider users community.

Among available approaches, OMF [4] 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, providing the infrastructural services needed to govern the system and its configuration. These software components support all the phases of an experiment life-cycle, from the provisioning of resources to the collection of experimental data.

OMF is composed of 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 in order to 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 the Publish-and-Subscribe (Pub-Sub) extension of the eXtensible Messaging and Presence Protocol (XMPP) [19]. 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 [20], ORBIT [21], and NICTA [22]), and has a large group of active users and a

community of 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 [23], 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 facility, but also enables running larger scale experiments using more resources than what a single facility could offer, and with wider variety of technologies.

The idea of testbed federation has emerged aiming at offering a seamless inter-connection between heterogeneous facilities, providing a common framework to provision the resources, and opening the testbed facility 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 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 its most visible and renowned reference implementation. It is successfully used to federate a variety of testbeds, including Planetlab [9], NITOS [20], Senslab [24] and Federica [25].

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* [26] and is responsible of performing all the slice instantiations. Also, it allows testbed aggregates 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 that is aware of a pre-configured set of other services, either Aggregate Manager or Slice Manager [27].

In order to enable OMF and SFA to work together and provision testbed resources, we need to add the functionnalities to list available nodes and reserve them for a user specified time slot. OMF treats resources as exclusive, preventing problems arising from concurrent users using the same wireless network cards or channels, which could affect the expected behaviour 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 listing and reserving testbed resources for a particular user-provided time slot.

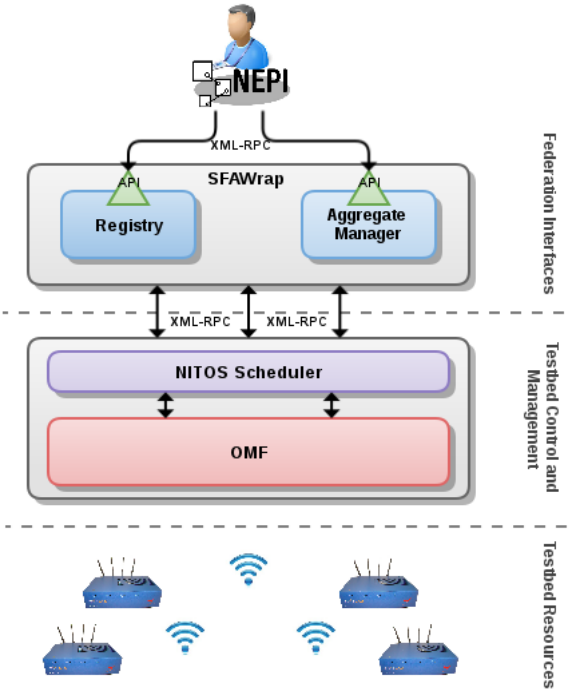


Figure 9: Overall architecture of our testbed control plane.

As no scheduler is included in OMF, we decided to use NITOS [28], a scheduler 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 the resources reservation is not enforced, but done 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

[1] A Slice is a group of resources dedicated to a particular experiment

a mechanism to monitor and control the resources access policies in order to prevent any conflict or malicious actions [29].

Furthermore, it implements a spectrum slicing scheme that enables better usage of wireless testbeds, by allowing to run several wireless experiments at the same time without interfering with each other [29].

Overall, integrating SFAWrap with the NITOS Scheduler and OMF was a very challenging work. Apart from the technical (engineering) aspects that bundles defining the communication interfaces and mechanisms, we had to address the specificities of our testbed. On the one hand, we extended the resource description scheme of SFAWrap to support wireless nodes and wireless channels. On the other hand, we introduced the time-based reservation of the resources into SFAWrap to enable the reservation of exclusive resources for specific time slots. Figure 9 illustrates the control plane of our testbed.

4.3. Testbed Infrastructure

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

Indeed, experiment's results obtained with real environment conditions are highly dependent on the network environment characteristics, and uncontrolled changes in the environment conditions can randomly affect the outcome of the experiment. Wireless networks present a particular challenge in this sense, because the communication channel is naturally exposed to uncontrolled electromagnetic interferences. To limit the effect of these undesirables interferences as well as get to closer to the reproducible experiments requirements, some insulation's technologies are necessary to insulate the wireless channel.

To achieve electromagnetic insulation of the testbed, different solutions exist and can be categorized in two types : insulation from outside and insulation from both inside and outside. The first one is commonly represented by a Faraday cage providing insulation from external electromagnetic interference only whereas the second one is usually illustrated by 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 [30] 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 anechoic chamber, use a thin layer of copper, of around 0.3 mm. The layer of copper covers all the surfaces to reduce the interference up to 100dB. Concerning the absorbent walls, foam pyramid absorbers will be used. The length of the foam pyramids determine the absorption of the reflected wave for a frequency range. The absorber have a size of 200mm in order to perform efficiently in Wi-Fi environments.

By building our testbed inside a fully anechoic chamber with the characteristics described above, we expect to provide adequate conditions for achieving, as much as possible, reproducible wireless 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 two (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 [20]). We decided to use about 40 Icarus NITOS nodes [31] 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 [32], which are mainly deployed in wireless testbeds, are not powerful enough for our needs (with their 1Ghz processor). Due to its high power consumption, power-over-ethernet cannot be used in our case. Therefore, Icarus nodes will use a Chassis Manager Card (CMC)to provide facility to force reboot and hard reboot. Icarus nodes also provide three wired interface slots, and two Ath9k wireless interfaces, that fit our requirements.

We are also investigating some others components to install in the testbed, such as "noise generator" to enable the possibility to control wireless network environment depending on the requirement of the user. In the future, we plan to investigate how to emulate mobility in the testbed, e.g., using an approach similar to iMinds [33, 34].

5. Conclusion

In this paper, we propose the Iterative Development of Network Protocols (IDEV) methodology that allows to combine in a same test environment fine-grained simulation, application-level emulation, protocol-level emulation, link-level emulation, and real experimentation. 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 describe the IDEV methodology through use cases and detail the different components that take part in the new framework. Then, we propose an IDEV-based network testbed architecture which supports automated scenario description, experimentation execution, testbed resources management/control, and federation with other testbeds. We believe that such a testbed can help researchers to evaluate and debug new network protocols and applications, as well as to allow reproducible performance results, which is particularly tricky in wireless scenarios.

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, in: OSDI02, USENIXASSOC, Boston, MA, 2002, pp. 255–270.

[2] D. Wischik, C. Raiciu, A. Greenhalgh, M. Handley, Design, implementation and evaluation of congestion control for multipath tcp, in: Proceedings of the 8th USENIX conference on Networked systems design and implementation, USENIX Association, 2011, pp. 8–8.

[3] M. Lacage, Outils D'expérimentation Pour la Recherche en Réseaux, 2010.
URL `http://books.google.fr/books?id=sNketwAACAAJ`

[4] T. Rakotoarivelo, M. Ott, G. Jourjon, I. Seskar, Omf: a control and management framework for networking testbeds, Operating Systems Review (2009) 54–59.

[5] 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, ACM, 2012, pp. 253–264.

[6] The ns-3 network simulator, http://www.nsnam.org/.

[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: 4th Symposium on Networked Systems Design and Implementation (NSDI 2007), April 11-13, 2007, Cambridge, Massachusetts, USA, Proceedings, USENIX, 2007.

[8] The emu network device, http://www.nsnam.org/docs/release/3.11/models/html/emu.html.

[9] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, M. Bowman, Planetlab: an overlay testbed for broad-coverage services, in: SIGCOMM '03, Vol. 33, ACM Press, New York, NY, USA, 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) 59–67.

[11] A. Bavier, N. Feamster, M. Huang, L. Peterson, J. Rexford, In vini veritas: realistic and controlled network experimentation, in: ACM SIGCOMM Computer Communication Review, Vol. 36, ACM, 2006, pp. 3–14.

[12] pcap and tcpdump, http://www.tcpdump.org/.

[13] M. Chesire, A. Wolman, G. M. Voelker, H. M. Levy, Measurement and analysis of a streaming media workload., in: USITS, Vol. 1, 2001, pp. 1–1.

[14] Usrp device, http://gnuradio.org/.

[15] ns-3/dce, http://www.nsnam.org/overview/projects/direct-code-execution/dce-1-1/.

[16] A. Quereilhac, C. Freire, M. Lacage, T. Turletti, W. Dabbous, Nepi: An integration framework for network experimentation, in: 19th International Conference on Software, Telecommunications and Computer Networks, SoftCOM '11, 2011, pp. 1–5.

[17] C. Freire, A. Quereilhac, T. Turletti, W. Dabbous, Automated deployment and customization of routing overlays on planetlab, in: 8th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, TRIDENTCOM '12, 2012.

[18] M. Ferrari, Netns: End of studies report ubinet master, http://hal.inria.fr/inria-00601848/fr/.

[19] Xep-0060: Publish-subscribe, http://xmpp.org/extensions/xep-0060.html.

[20] Network implementation testbed laboratory, http://nitlab.inf.uth.gr/NITlab/.

[21] Orbit website, http://www.orbit-lab.org/.

[22] Nicta website, http://www.nicta.com.au/research/projects/tempo.

[23] X. Lin, Y. Mao, F. Li, R. Ricci, Designing a federated testbed as a distributed system, in: Proceedings of the 4th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud), 2012, to appear.

[24] Senslab website, http://www.senslab.info/.

[25] Federica website, http://www.fp7-federica.eu/.

[26] Geni aggregate manager api, http://groups.geni.net/geni/wiki/GAPI_AM_API.

[27] L. Peterson, S. Sevinc, S. Baker, T. Mack, R. Moran, F. Ahmed, Planetlab implementation of the slice-based facility architecture, www.cs.princeton.edu/ llp/geniwrapper.pdf (Jun. 2009).

[28] Nitos testbed scheduler, http://nitlab.inf.uth.gr/NITlab/index.php/scheduler.

[29] A.-C. Anadiotis, A. Apostolaras, D. Syrivelis, T. Korakis, L. Tassiulas, L. Rodriguez, M. Ott, A new slicing scheme for efficient use of wireless testbeds, in: Proceedings of the 4th ACM international workshop on Experimental evaluation and characterization, WINTECH '09, ACM, New York, NY, USA, 2009, pp. 83–84.

[30] K. Liu, Anechoic chambers for evaluating and validating telecommunication equipment, Safety and EMC Magazine.

[31] Icarus node description,
http://nitlab.inf.uth.gr/NITlab/index.php/testbed/hardware/wireless-nodes/icarus-nodes.

[32] Orbit node,
http://groups.geni.net/geni/wiki/GeniAggregate/OrbitWirelessTestbed.

[33] iminds, http://www.crew-project.eu/wilabt.

[34] 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, Springer, 2012, pp. 7–8.

Author Biographies

Young-Hwan Kim received B.S., M.S., and Ph. D. degrees in computer science from the Yonsei University, Wonju, Korea, in 2002, 2004, and 2011, respectively. Since 2012, he has been with the Planéte team at INRIA, Shopia-Antipolis, France, as a post-doctoral researcher. His current research interests include quality of service in wireless LANs, wireless sensor networks, video streaming over the wireless networks, and performance evaluation through computer simulations.

Alina Quereilhac holds a degree in informatics engineering from the University of Buenos Aires, Argentina, in 2008. She is currently a Ph. D student at the University of Nice-Sophia Antipolis, working in the Planéte team at INRIA, on the topic of unified evaluation environments for networking protocols and applications on simulators and testbeds.

Mohamed Amine Larabi holds an engineering degree in computer systems from Ecole nationale Supérieure d'Informatique (ESI), Algiers, Algeria, in 2009. He is currently an expert engineer in the Planéte team at INRIA, working on the federation of network testbeds.

Julien Tribino holds a master degree in computer science from the Bordeaux Graduate School of Engineering (ENSEIRB-MATMECA), France, in 2011, with a speciality in the network and distributed systems. He is currently working as an engineer at INRIA, Shopia-Antipolis, France. His main interests are the wireless network, the distributed systems, and the network security.

Thierry Parmentelat is a senior research engineer at INRIA. Prior to joining INRIA, he has held various positions within the software industry, has been CTO at Centile, a VOIP company, and formerly he has designed and implemented critical systems for industrial sectors, like nuclear energy and aerospace. His areas of interest include experimental testbeds, networking, and programming languages. He graduated from École Polytechnique in 1984, and received his PhD in 1991.

Thierry Turletti received the M.S. (1990) and the Ph.D. (1995) degrees in computer science from the University of Nice - Sophia Antipolis, France. He has done his PhD studies in the RODEO group at INRIA where he

designed the Inria Videoconferencing System (IVS). During the year 1995-96, he was a postdoctoral fellow in the Telemedia, Networks and Systems group at LCS, MIT and worked in the area of Software Defined Radio (SDR). He is currently a senior research scientist at the Planéte project-team at INRIA Sophia Antipolis. His current research interests include information centric network architectures, trustable network evaluation platforms and wireless networking. Dr. Turletti has been serving on the Editorial Boards of the following journals: Wireless Communications and Mobile Computing (WCMC) [2001-2010], Wireless Networks (WINET) [since 2005] and Advance on Multimedia (AM) [since 2007].

Walid Dabbous is a senior researcher at INRIA and professor at the Ecole Polytechnique. His research interests include: Future Internet Architecture and Protocols, Networking Experimental Platforms and Simulators, Experimental Methodology for Networking Protocols, Securing Internet coordinate systems, Large scale virtual environments, Large scale reliable multicast protocols, Group communications, Internet Satellite Networking, Quality of Service support, Buffer management mechanisms in routers, Flexible protocol architecture, High performance communication protocols, Unicast and Multicast Congestion control, Audio and video conferencing over the Internet. He graduated from the Faculty of Engineering of the Lebanese University in Beirut in 1986 (Electrical Engineering Department). He obtained his DEA and his Doctorat d'Université from the University of Paris XI in 1987 and 1991 respectively. He joined the RODEO Team within INRIA in 1987. He is a staff researcher at INRIA since 1991, and leader of the RODEO (then Planéte) team since 1996.

Author Photos

(a) Young-Hwan Kim

(b) Alina Quereilhac

(c) Mohamed Amine Larabi

(d) Julien Tribino

(e) Thierry Parmentelat

(f) Thierry Turletti

(g) Walid Dabbous