OpenAirInterface Large-Scale Wireless Emulation Platform and Methodology *

Ben Romdhanne Bilel; Nikaein Navid; Knopp Raymond; Bonnet Christian
Eurecom
2229, route des Cretes
06904 Sophia Antipolis, France
{nikaeinn,romdhan,knopp,bonnet}@eurecom.fr

ABSTRACT

The OpenAirInterface emulation platform is an integrated tool allowing large-scale networking experimentation applicable to both evolving cellular and adhoc/mesh topologies. The platform in built to represent a realistic system in a controlled and real-time environment, which interacts with the external elements. The platform has a dual objective of performance evaluation of application and protocols as well as their testing and validation.

Categories and Subject Descriptors

H.4 [**Techniques for network measurement, simulation, and emulation**]:

General Terms

LTE, LTE-A, WMN, Emulation, methodology, experimentation

Keywords

Large Scale, Emulation, LTE, Evaluation

1. INTRODUCTION

The next generation wireless systems, protocols, and applications applicable to evolving cellular and adhoc/mesh networks are becoming complex and hence their performance evaluation is difficult and in some cases unreliable. This makes the experimental approach necessary in order to validate and compare such systems and protocols. The question that arises in this regard is that how such an experimental approach could be applied to a large scale systems. Furthermore, wireless systems are subjected to stochastic factors from environment and mobility, which need to be controlled or assessed in order to obtain conclusive results. To achieve a meaningful experiment the following properties are required (i) *reproducibility:* repeat the experiment in a controlled environment, (ii) *scalability:* run large scale networking experimentations, and (iii) *applicability:* represent a set of realistic systems, protocols, and scenarios [3] [6].

There exist three main approaches when evaluating the performance: simulation, emulation, and real testbed. Simulation is typically done in controlled environment, where some or all part of the system and network stack are modeled. Thus, the execution environment is not that of the real system and the interactions with external entities are limited (e.g. real application). This makes simulation reproducible and scalable but not applicable as the use of models may deviate results and system behavior, and can hide important issues when the software is implemented and deployed on a large scale in a real environment. At the other end, real testbed provides a semi-controlled environment, where almost all the elements of the system are real. The execution environment is also real and open to external entities with their IO stream making this approach applicable. Nevertheless, real testbed is expensive and not scalable and the measurements produced on them are hard to predict and reproduce. The best approach would be that of a real system and network stack (e.g. Linux or BSD) in a controlled environment, where a given experiment could be reproduced to verify the true difference between two different solutions. However, the decision on which element of the protocol stack is real or modeled depends on the use case and purpose of the experiments. The OpenAirInterface (OAI) emulation platform described here intends exactly to provide such an environment [4]. Furthermore, the main difference between the methodology used with respect to existing open-source simulation/emulation tools such as NS [2], is firstly that it is built with a real-time framework in mind, using the open-source real-time application interface (RTAI) extension to Linux [5], and secondly that it is part of a validation chain for a real protocol implementation.

This paper is organized as follows. Section 2 provides an overview of OAI emulation platform, its workflow and architecture. Section 3 presents a concrete example scenario and some result. Finally some conclusion and future directions are given.

2. OAI EMULATION PLATFORM

The OAI emulation platform is an open-source software modem and layer 2/3 protocol stack solutions for wireless network experimentation written in C targeting both real-time and non real-time operations. The current development targets generic Linux-based hardware environment ranging

*This work has been presented at ACM MSWIM 2011 demo session

PM^{2}HW^{2}N'11, October 31, 2011, Miami, Florida, USA.

from a simple PC to sophisticated PC cluster or even a GPU workstation.

The software platform currently aligns its air-interface development with the evolving LTE standard but provides extensions for adhoc/mesh networking, particularly in the MAC and Layer 3 protocol stack. This emulator allows protocol and application performance evaluation and validation as well as system testing. OAI provides a complete wireless protocol stack and radio hardware and implements the PHY, MAC, RLC(Radio Link Control), RRC(Radio Resource Control) layers as well as providing a IPv4/IPv6 network device interface under Linux[1]. It can be seen as a mock standard which retains the salient features of a real radio system, without all the required mechanisms one would find in a standard used in deployment of commercial networks. The aim is to study innovative techniques such as cooperative protocols, relaying scheme, interference management in a set of conventional (e.g. VoIP) and emerging applications (e.g. machine-to-machine and multiplayer gaming).

2.1 Experiment Design Workflow

A sequential experiment workflow, where the output of each step will be the input of the next, is employed to allow an experiment to be reproduced. Five consecutive steps are defined: *scenario description, configuration, execution, monitoring, analysis*, where each step is split into several sub-steps as explained in the following subsections (see Figure 1.

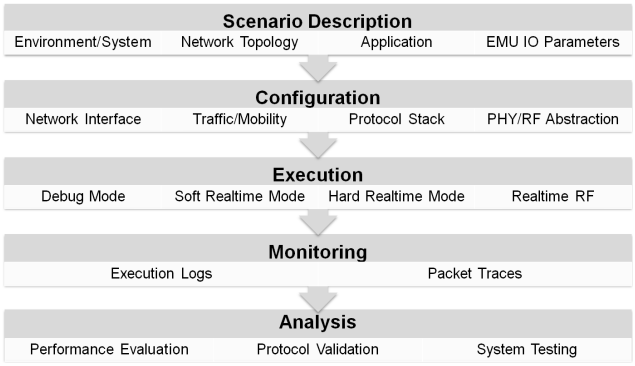

Figure 1: Experiment Design Workflow

2.1.1 Scenario Description

This step builds a complete xml layout of an experiment. This step is splitted into four sub-steps: (i) *system/envirnment*, where system (e.g. bandwidth, frequency, antenna) and environement (e.g. pathloss and channel models) parameters are defined; (ii) *network topology*, where network area, network topology (i.e. cellular, mesh), nodes' type, initial distribution, and mobility model (e.g. static, random way point, grid) are set; (iii) *application*, where real application and/or emulated traffic pattern in terms of packet size and inter-departure time are defined; (iv) *EMU IO Parameters*, where supervised parameters (e.g. emulation time, seed, performance metrics) and analysis method (e.g. protocol PDUs and operation) are set.

2.1.2 Configuration

This step defines a sequence of components' initialization based on the scenario description. It includes four sub-steps: (i) *network interface*, where the OAI IP interface is configured, (ii) *traffic and mobility*, where traffic pattern and mobility model parameters are set, (iii) *protocol stack*, where protocols are configured given the network topology and *PHY abstraction*, where a channel model prediciting the modem performance is configure.

2.1.3 Execution

This step defines the execution environment for the emulator in order to synchronize the emulated nodes and run the experimentation. It includes four execution modes: (i) *debug mode*, where the emulation is executed in user space without any Linux IP connectivity, (ii) *soft real-time mode*, where the debug mode is interconnected with the Linux IP protocol stack and calibrated to respect the layer 2 frame timing on average, (iii) *hard real-time mode*, where the emulation is executed in RTAI kernel with Linux IP protocol stack respecting layer 2 frame timing strictly, and (iv) *real-time RF mode*, where hard real-time mode is extended with the RF equipment interconnected with an attenuator.

2.1.4 Monitoring

This step defines how the experiment is monitored (passive and/or active). It includes: (i)*execution logs*, where experiment traces and logs are collected, labeled and archived, (ii) *packet traces*, where protocol signaling is captured and stored during the experiment.

2.1.5 Analysis

This step processes raw data and produces results and statistics. It includes three -non exclusive- analysis objectives: (i)*performances evaluation*, where the key performance indicators are measured and evaluated, (ii) *protocol validation*, where the protocol control- and user-plane signaling are validated versus protocol specification, and (iii)*system testing*, where the system as a whole is analyzed and tested.

2.2 Architectural Design Choices

To increase both scalability and applicability of an emulation, OAI applies five architectural design choices: (i) *real protocol stack*, where protocols are implemented as in a real system (not modeled), (ii) *protocol virtualization*, where the emulated network nodes share the same operating system instance and Linux IP protocol stack within the same physical machine, (iii) *transparent emulated data flow*, where emulated data are exchanged either via a direct memory transfer when they are part of the same physical machine or via multicast IP over Ethernet when they are in different machines, (iii) *parallel processing*, where virtualized protocol instances and CPU expensive functions are executed in separate threads (iv) *end-to-end validation*, where real application client/server are attached on the top of some emulated nodes and the remaining traffic pattern and mobility model will be generated automatically as would be the behavior of the real application at the client and server sides.

2.2.1 Real Protocol Stack

OAI provides a complete protocol stack for cellular and mesh network topologies applicable to both emulation and real-time RF platforms as shown in Figure 2. At the access layer, it implements the full or abstracted PHY, MAC,

RLC (radio link control); RRC (radio resource control) layers and provides an IPv4/IPv6 network device interface under Linux. The abstracted PHY is used to enable fast emulation of complex network by predicting the modem performance in terms of BLER (block error rate). It injects error patterns for each transport channel at the receiver of each emulated node based on the wideband SINR, network topology, and propagation model or real channel traces. At the network layer, OAI implements RRM (radio resource management), routing, multicasting, topology control, and proxy mobile IP.
The software implements the standard network socket interface for IP layer and a generic request/indicate interface among other layers making the design highly modular. The generic interface also provides a loopback interface at each (sub-)layer for testing and validation purpose.

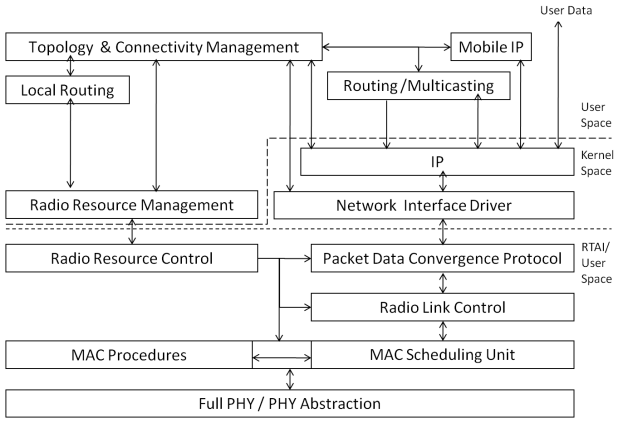

Figure 2: Protocol Stack

2.2.2 Protocol Virtualization

OAI provides virtualization of the network nodes within the same physical machine to increase the scalability of the emulation platform (see Figure 3). Protocol virtualization consists of sharing the same operating system instance and Linux IP protocol stack for independent emulated node instances. It allows networks nodes to coexist in the same execution environment. In some cases, the use of OS virtualization tools such as UML may be needed for the development and performance evaluation of sophisticated layer 3 protocols. Note that, protocol virtualization offers the same functional properties (i.e. services) and the same non functional properties (i.e. performances) than that of a real protocol.

2.2.3 Transparent Emulated Data Flow

To increase scalability and allow experimentation of a complex network topology and architecture, two emulated data flows may coexist between emulated nodes as shown in Figure 3. Either nodes communicate via direct memory transfer or via IP multicast (over Ethernet) depending on whether they are part of the same physical machine or not. From the point of view of the protocol stack, the data flow is transparent and that the network connectivity is independent from the node distribution on a physical machine.

2.2.4 Parallel Processing

To achieve scalability and exploit the current multi-core hardware architecture, OAI implements two types of parallelisms: logical and useful. Logical parallelism applied to the virtualized protocol stack in order to separate emulated nodes from each other as in a real network, while useful parallelism is applied to PHY abstraction and channel modeling to isolate the CPU expensive functions. Indeed, parallelization feature need to redesign the software which present an important development overhead.

2.2.5 Flexible End-to-End Validation

Real applications can be attached on the top of some emulated nodes via the OAI network interface, which provides Linux IP interconnection with QoS classification on layer 2 resources. Such an application may interact with an external application through an emulated or true core network (e.g. WANEM [7]). The remaining traffic pattern and mobility model will be automatically generated as would be the behavior of the real application at the client and server sides.

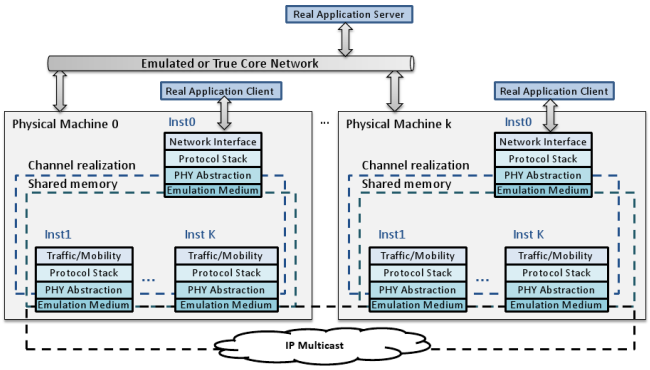

Figure 3: E2E Validation Architecture

3. DEMONSTRATION SCENARIO

The demonstration setup is composed of a simple laptop equipped with a quad-core CPU running OAI emulator and protocol stack using Linux on Ubuntu 10.04. This configuration allows soft real-time execution mode. We consider a simple cellular network composing of one eNB (enhanced NodeB) and up to three UEs (User Equipment). We make use OAI HMI (Human Machine Interface) to describe experiment scenario and analyze the output XML file. At this step, we present the traffic mobility operation. All experimentation will be run using OAI network driver and each emulated node will have its own network interface. We plan to run 2 experimentations classified as follows:

- In the first experiment, we make use of PHY abstraction module (upper path in figure 5) and increase the number of UEs from 1 to 3.
- In the second, we make use of the full PHY layer implementation (lower path in figure 5) and keep only 1 UE.

The traffic pattern will be based on a combination of simple video streaming (VLC) in the first experiment and a traffic generator in the second one. The traffic generator emulates a sensory data characterized by a small packets size and inter-departure time.

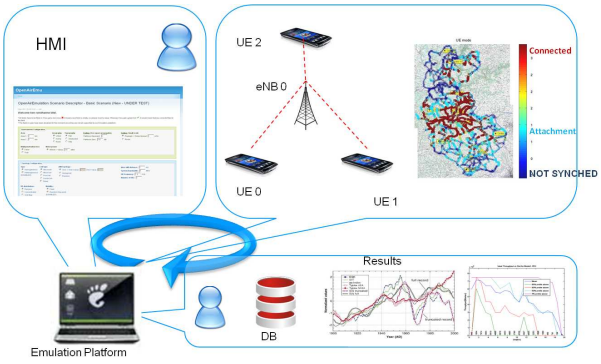

Figure 4: Scenario Design

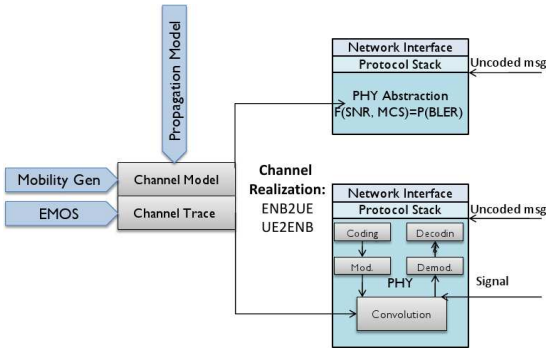

Figure 5: PHY abstraction models

3.1 Results

We will make use of Wireshark to demonstrate protocols performances under different conditions. In the following tqble we only present results related to the performance of the platform itself related to the two above-mentioned experiments. For this purpose, we use Gprof -a profiling software used to collect and arranges statistics - to evaluate the performance of each software block of the emulation platform and measure the scalability.

In the first experiment where we use PHY abstraction, we

Table 1: OAI Profiling results

Experiment 1

Time(s)	Channel	PHY abst	L2 Proto	OS	Total
1ue	3.09	0.4	0.3	0.8	4.58
2ue	6.38	0.49	0.52	1.34	8.73
3ue	9.03	0.4	0.78	1.85	12.03

Experiment 2

Time(s)	Channel	PHY	L2 Proto	OS	Total
1ue	230	39.35	8.86	0.16	278.27
2ue	4675.13	792.15	119.07	0.3	5587.59
3ue	8690.38	1134.90	300.59	0.47	10126.29

note that emulation time increases linearly as a function of emulated nodes of number(ascertainment validated until 9 UEs). In particular, channel modeling represents up to 80% of total CPU load, while protocol stack does not exceed 20% of total CPU load.

In the second experiment, we make use of full PHY implementation, and we notice that the execution time did not increase linearly. Moreover, channel modeling function represents again 80% of the execution time followed by PHY functions with 12%, and Protocol stack with 8%.

These results show that the main performance bottleneck of the emulation resides in channel modeling block specially when full PHY is used. Then, the trivial optimization approach may be applying useful parallelism technique in this software bloc.

4. CONCLUSION

In this paper, we present the OpenAirInterface emulation platform and methodology. The methodology aims at providing a repeatable and realistic wireless network experiment for a large scale network. The platform implements a sequential workflow allowing experiment reproducibility. Further, OAI propose a user-friendly workspace which simplifies its utilization. Furthermore, OAI applies four architectural design features in order to increase scalability and applicability: real protocol stack, protocol virtualization, transparent emulated data flow and parallel processing.

In the proposed scenario, we demonstrate the importance of the sequential workflow and the impact of suggested design features. Our profiling results show that the emulation bottleneck resides in channel modeling bloc.

In futures works we aims to resolve two challenges: first the applicability challenge using real channels trace and second the scalability challenge by applying useful parallelism based on GPGPU techniques.

5. ACKNOWLEDGMENT

This paper describes work undertaken in the context of the LOLA and CONECT project. The research leading to these results has received funding from the European Community's Seventh Framework Programme under grant agreement nÂř 248993 and nÂř257616.

6. REFERENCES

[1] D. C. F. F. R. K. Hicham Anouar, Christian Bonnet. An overview of openairinterface wireless network emulation methodology. *SIGMETRICS*, 2008.

[2] D. Mahrenholz and S. Ivanov. Real-time network emulation with ns-2. In *8th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, 2004.

[3] B. B. R. Navid NIKAEIN. Wireless network experimentation and validation: A survey. RR-11-256:1–25, JUN 2011.

[4] OpenAirInterface. www.openairinterface.org.

[5] Real Time Application Interface. www.rtai.org.

[6] R. G. T Staub. Virtualmesh: an emulation framework for wireless mesh networks in omnet++. *SIMULATION*, 2010.

[7] The Wide Area Network emulator. wanem.sourceforge.net.