Ann. Telecommun. (2012) 67:247–255
DOI 10.1007/s12243-011-0268-5

When should I use network emulation?

Emmanuel Lochin · Tanguy Pérennou · Laurent Dairaine

Received: 4 January 2011 / Accepted: 14 June 2011 / Published online: 6 July 2011

Abstract The design and development of a complex system requires an adequate methodology and efficient instrumental support in order to early detect and correct anomalies in the functional and non-functional properties of the tested protocols. Among the various tools used to provide experimental support for such developments, network emulation relies on real-time production of impairments on real traffic according to a communication model, either realistically or not. This paper aims at simply presenting to newcomers in network emulation (students, engineers, etc.) basic principles and practices illustrated with a few commonly used tools. The motivation behind is to fill a gap in terms of introductory and pragmatic papers in this domain. The study particularly considers centralized approaches, allowing cheap and easy implementation in the context of research labs or industrial developments. In addition, an architectural model for emulation systems is proposed, defining three complementary levels, namely hardware, impairment, and model levels. With the help of this architectural framework, various existing tools are situated and described. Various approaches for modeling the emulation actions are studied, such as impairment-based scenarios and virtual architectures, real-time discrete simulation, and trace-based systems. Those modeling approaches are described and compared in terms of services, and we study their ability to respond to various designer needs to assess when emulation is needed.

Keywords Experimentation · Emulation · Protocol · Internet

E. Lochin (✉) · T. Pérennou
CNRS; LAAS, 7 avenue du colonel Roche,
31077 Toulouse, France
e-mail: emmanuel.lochin@isae.fr

E. Lochin · T. Pérennou · L. Dairaine
Université de Toulouse, UPS, INSA, INP, ISAE; LAAS,
31077 Toulouse, France

1 Introduction

Designing and developing communication protocols and real-time systems is a complex process where various actors participate in different phases, having only a partial vision of the whole system. The experiment phase, which eventually provides a global vision, is a mandatory step in research and development process of distributed applications and communication protocols. In this context, three classical ways to achieve experimentation are commonly used: simulation, live testing, and more recently emulation.

Simulation, particularly event-driven simulation, is a classical way to achieve economical and fast protocol experimentation. It relies on an ad hoc model to work with and it uses a logical event-driven technique to run the experiment. The use of modeling techniques simplifies the studied problem by concentrating on the most critical issues. Indeed, network simulators are essential to provide a proof of concept prior to protocol development. Nevertheless, those tools (based on a virtual clock) cannot replace practical protocol evaluation that quantifies implementations' overhead during real-time operation. Eventually, to realize a

real-time evaluation, only two solutions are left: live testing and network emulation. For instance, network simulation has been used to design the TFRC protocol internal mechanisms [24] (the TCP-Friendly Rate Control protocol is a rate-based congestion control mechanism for unicast flows operating in a best-effort Internet environment); then, a user-level prototype has been realized [37] to quantify the processing overhead related to the inherent implementation. Throughout the remainder of this paper, we use the TFRC case as a running example to help understand certain concepts presented.

In live testing, evaluations are driven with real implementations. The fundamental way to do the experiment is by using real technology for the underlying networking environment. This real environment can be the target network or an ad hoc testbed involving real equipments. Nevertheless, this approach is considered to be very expensive and inflexible to evaluate all aspects of the protocol being tested.

Emulation is considered to be at the cross-road between simulation and live testing. This approach consists in executing and measuring real protocols and application implementations over a certain network where part of the communication architecture is simulated in real time. The aim of emulation is to allow a distributed software to run either in realistic conditions (e.g., over a satellite network) or specific conditions (e.g., when specific packets are dropped such as SYN packets in the TCP case).

This paper introduces an overview of existing network emulation approaches. We particularly focus on the study of centralized approaches, allowing simple implementation in the context of research labs or industrial development centers. In addition, we propose a general architectural model that allows various emulation approaches to be presented and situated in the model. These emulation approaches include mainly impairment scenarios and virtual architectures. Furthermore, a comparison of these approaches and a set of criteria considered as requirements for emulation systems will be proposed.

This paper is organized as follows: Section 2 situates emulation among various experimentation approaches. Section 3 presents main emulation requirements. Additionally, Section 4 discusses network emulation architecture based on three complementary levels. The outline of main emulation approaches, as well as their positions in the architectural model proposed will be discussed in Section 5. Finally, some concluding remarks are given in Section 7.

2 Network experimentation approaches

Before diving in the world of emulation, we first present the common experimental approaches used in research labs and industrial development.

2.1 Simulation

As discussed above, simulation is a very effective and efficient way to experiment with protocols. Network simulation typically utilizes ad hoc model and logical event-driven techniques. Classical tools such as ns-2 [10] or OPNET [14] provides a core simulation engine, as well as a large set of protocol models. These simulation tools allow experiments to be done without high costs involvement. The modeling techniques used in the simulators allow the studied problems to be simplified by concentrating on the most important issues. Furthermore, simulation tools do not operate in real time. Therefore, depending on the model complexity, it is possible to either simulate a logical hour in few real-time milliseconds or a logical second in several real-time days. This characteristic reflects both the benefit and weakness of simulation tools. Due to this attribute, it is unfeasible for simulation tools to implement systems involving man-in-the-loop. Most simulation tools do not allow to test real-time implementations but only models (even the most innovative and sophisticated one).

2.2 Simulation shortcomings

As already emphasized in the introduction, network simulators are essential to provide a proof of concept prior to protocol development but cannot replace practical protocol evaluation. To pursue with the TFRC example previously introduced in Section 1, the concept of the rate-based equation has been validated within ns-2 simulator while the feasibility of the implementation has been evaluated through real-time experiments. Even though the core algorithm developed within ns-2 in C++ has been reused inside the kernel implementation, most of the data structure, message exchange, and protocol framework had to be written from scratch. This additional code has to be evaluated too.

Furthermore, it is important to ensure that the services and performances offered by the simulation model are consistent with the real experimental implementation of the protocol.

2.3 Live experimentation

Another conventional method to test and debug distributed software during the implementation stage is to use real hardware and/or software components. The software can be tested either on a real target network or on an ad hoc testbed using real equipment. However, this approach is particularly expensive in the context of wide area networks, especially when using specific technology such as satellite network. The cost inefficiency of this method does not involve only the technology cost but also the distributed man-in-the-loop manipulations and synchronization required. Moreover, it is sometimes impossible to use this approach simply because the new technology support is not yet validated or available, e.g., when developing an application over a new satellite transmission technology that is not yet operating. This method also suffers from the inherent discrepancies between a particular test network and the much broader range of network imperfections that will be encountered by the software users.

Using real technology on target operational network has been widely deployed. An example of this scheme is well illustrated by PlanetLab [32]. PlanetLab is a distributed platform that alleviates experiments management, offering a way to use a very large set of hosts over the Internet. However, the purpose of PlanetLab is to use Internet as a testbed and not to control the network experimentation conditions. As a result, PlanetLab does not target reproducibility and is, thus, more efficiently used for metrology experiments.

2.4 Network emulation

Since several years, progresses in high speed processing and networking have allowed the rapid development of network emulators, such as Dummynet [34], NIST Net [16]. Network emulation is a weighted combination of real technology and simulation. It is used to achieve experiments using both real protocol implementations and network models. Basically, this allows the creation of a controlled communication environment. This communication environment can produce specific target behaviors in terms of quality of service. The objective of emulation tools is to reproduce a real underlying network behavior, such as configurable wired [39] or wireless [17, 41] topologies. Additionally, emulation aims at providing "artificial impairments" on the network to test particularities of the experimented protocol. These impairments include loosing specific packets, reducing the network bandwidth with a specific timing or introducing delay over the network. Emulation is particularly useful in the debugging and testing phase of a system.

3 When do I need emulation?

You need emulation to assess the performance of an end-to-end system. Although you can use emulation at any layer of the OSI model, in the present paper, we focus on network emulation which is a combination of real technology (application and communication stack above link-level) and simulation of the behavior of the link and physical levels.

Let's assume you want to assess the performance of the TFRC transport protocol. You might be interested in validating the use of your implementation over several types of terminal e.g., mobile phone, PDA, laptop, server, etc.) and compare whether the impact of TFRC internal algorithms behave similarly in various network conditions. In order to drive this test, you must evaluate as a non-exhaustive list: the memory footprint, CPU usage, and packet processing overhead to identify potential limits and propose implementation and algorithmic improvements. These metrics are not available in a simulation context. Testing TFRC with various bandwidth size in a real setup would involve the use of several different network setups (i.e., several testbeds with different cards on different hosts) while emulation provides an easy way to set the bandwidth of a link. Anyway, network emulation is the most practical scheme to obtain trustable metrics since you change only one parameter of your experimental setup.

In order to assess Quality of Service (QoS), the overall performances obtained from an end-to-end protocol are mainly dependent on external factors such as underlying technologies (e.g., RSVP establishment path, Service Level Agreement with a DiffServ network, etc.), interconnection topologies, current network traffic and so on. Different types of QoS can also be offered by the underlying network. For example, IP network service could offer a communication channel ranging from perfect (minimum delay, high bandwidth and no packet loss, as in a gigabit LAN) to unsatisfactory (high delay, low bandwidth, and high packet loss rate (PLR), as in a noisy satellite network), depending on the underlying protocols and many other external factors. This leads to a large set of possibilities in the protocol experiments that can be created. The different types of end-to-end

QoS that can be produced by the underlying experiment framework can focus on:

- *Artificial QoS*: the experiment framework provides a way to evaluate the protocol over specific QoS conditions, not imperatively related to any technology or realistic conditions. Artificial QoS allows the user to test and focus on its experimental protocol in target QoS conditions. This can be considered as a form of unit testing. Furthermore, the aim of this method is to point out errors or bugs that are difficult to observe in a non-controlled environment where they rarely happen. This can be used, for instance, at the transport level to study the impact of various packet drops in a TCP connection (e.g., SYN/ACK packets [22], etc.).
- *Realistic QoS*: the experiment framework provides a way to reproduce the behavior of some specific network architecture as accurately as possible. This type of experiment allows the user to evaluate the protocol over an existing network or inter-network without using a real testbed and all related technologies (e.g., a wireless network, a satellite network, an Ethernet gigabit network, or any interconnection of such technologies).

Generally, the following set of impairments are commonly at least supported by almost all emulators systems: round-trip time delay, jitter, packet loss rate, and bandwidth size.

Today, there are several emulation platforms freely available on the Internet, either remotely accessible (e.g., EmuLab [36] and Orbit [29, 33]) or for download and local installation (e.g., IMUNES [40], Netem [25], Dummynet [34], and KauNet [23]). We strongly believe it would not be appropriate to simply list and detail all these proposals. Instead, we propose in the following section an architectural model where essential features are highlighted.

4 Network emulation architectural model

Network emulation systems are based on various conceptual levels as illustrated in Fig. 1. In this figure, we split an emulation system into three complementary levels, denoted model level, impairment level, and hardware level. Each of these levels will be discussed in more details. Note that the user system is not considered to be a part of the emulation system. It includes the system under test, for instance a protocol or a distributed application to be evaluated or demonstrated as well as traffic sources and sinks.

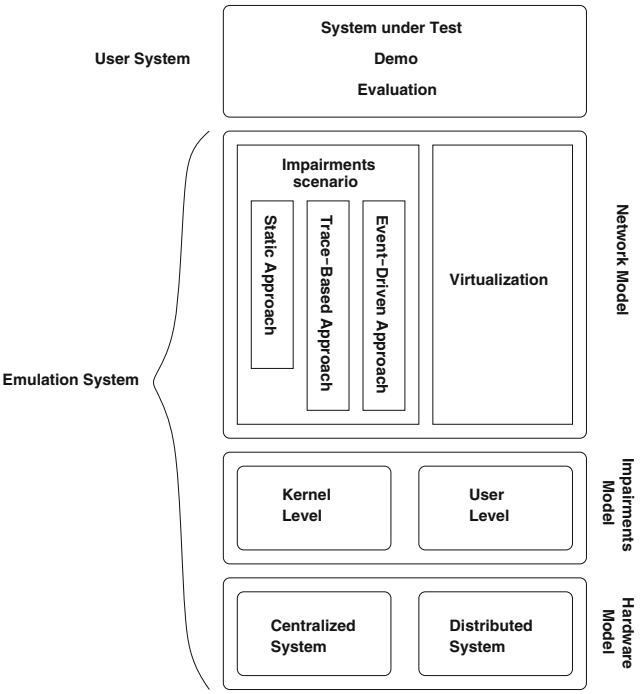

Fig. 1 Architectural model for emulation systems

4.1 Hardware level

The lowest layer of the proposed architecture, namely *hardware layer*, represents the physical devices really used by the emulation system. These devices comprise the real end-systems, the real network links that interconnect them and possibly, network components such as switches or routers. The virtual resources of the rest of the emulation system and the user system are mapped on those real resources, e.g., several virtual end-systems can reside on a single real computer. It is crucial to understand that hardware level is not necessarily composed by the technologies associated to the emulated network conditions. For instance, emulating a satellite link to evaluate the performance of the TFRC protocol can be roughly done over a few desktop stations interconnected with ethernet links by setting appropriate PLR and delay on the resulting emulated link (see Section 5.1.1).

The emulation system itself can be based on either a centralized system or a distributed system. In a centralized emulation system, we only use one computer to host the sender(s), the receiver(s), the intermediate node(s), and to manage all the impairments which define the experiment; while a distributed emulation system uses several computers to realize the same task. As an example, the IMUNES [40] system falls in the first category while Dummynet [34] or EmuLab [36] belong to the second one.

The main advantage of using distributed rather than centralized emulation system is the computation efficiency. However, this can also be considered as a disadvantage as it requires more physical resources and then, is much more complex to manage and administrate. For example, an EmuLab-like testbed requires at least five computers: a sender, a receiver, a core emulator, and two computers used to emulate both links in order to drive an experiment. This raises the problem of time synchronization of all machines that can be solved by using the NTP protocol [30]. Despite the use of NTP and in the context of delay estimations, computers can experience clock drift that might compromise the measurements. Although a distributed emulation system is greedy in terms of resources, it remains more appropriate to estimate overall resource usage consumed by the protocol itself as it isolates the protocol under test from the emulation system. The advantage of a centralized emulation system is that it shares the same clock for all its components and is inherently synchronized.

4.2 Impairment level

The *impairment level* provides a mean to introduce impairments over the exchanged packet flows. The impairment system is a center piece of the whole emulation because the real target network conditions are driven by the impairment system. The accuracy of the emulation is deeply associated to the capacity of impairment systems to process the packets in time and without introducing any other impairment than those specified in the upper level. For instance, the impairment processing overhead might bias the packets processing time estimation.

An impairment can be introduced at either the kernel level or the user level. An example of emulator that introduces impairment at the kernel level is Dummynet [34]. Dummynet intercepts packets at the IP forwarding level by implementing a queue (named pipe by Dummynet API) able to introduce impairments on the enqueued packets. Dummynet is configured through the FreeBSD firewall API where each pipe is set up as a simple forwarding rule. Another similar tool implemented inside the GNU/Linux kernel is NIST Net [16]. While Dummynet employs sophisticated queuing models for bandwidth modeling, NIST Net includes delay models of much statistical sophistication. Indeed, NIST Net is able to implement a varying delay scenario according to a given distribution while Dummynet uses a static delay because of the use of a queue. Both emulators cover complementary needs.

Finally, an example of user-level impairment is ONE [11]. ONE provides similar capabilities as Dummynet at the user level. However, the clock timer resolution is a function of the kernel configuration and in general it ranges from 1 to 10 ms. Indeed, several system scheduler runs at a default 100 Hz, meaning times based on normal system calls cannot be more precise than 10 ms. As a result, a user-level emulation cannot be as accurate as a kernel level one which gets a granularity close to the nanosecond. However, this approach is simple to install and adapted for many simple educational purposes.

4.3 Network model level

The *model level* defines two ways to control the emulation behavior. A *user impairment scenario* consists of an explicit list of impairment events while a *virtual network architecture* generates implicit impairment events based on the virtual topology, equipments, link characteristics, communication and routing protocols. Both models will be discussed in more details in the next section.

5 Emulation approaches

5.1 Impairment scenario models

There are various types of scenarios. They can be classified as impairment scenario models as described throughout the rest of this section.

5.1.1 Static approach

In a static approach, every parameter remains constant throughout the experiment. Therefore, the static settings need to be configured before the experiment is conducted. It does not describe the real network very accurately since the behavior of real network changes all over time. However, it is sufficient to reproduce pragmatic cases of artificial quality of service (e.g., bounded delay which characterizes specific network such as satellite link). The parameters that can be defined statically include delay, packet loss rate, bit error rate, packet reordering, etc. This emulation model is usually useful to test all the possibilities of a product or to compare it to other already existing products. This is the basic behavior of emulator such as Dummynet [34] which is mostly used in this way.

5.1.2 Event-driven approach

The key idea in event-driven [19] approaches is to apply impairments according to events. The most commonly used events are clock ticks (time-driven approach), but other events can be used, such as packet numbers, specific conditions observed on the traffic, or purely random occurrences. The event-driven approach is very useful to schematically represent a general behavior. The tester will be able to validate the product under several conditions and to compare it with other solutions. This approach has been used in various emulation tools. For example, Net Shaper [26] uses time oriented emulation. In Net Shaper, a daemon is executed and that daemon would wait for the new model to be applied to the emulation processor. The daemon is able to successfully receive and process up to 1,000 messages/s.

In the case of clock tick events, all impairments are triggered at user-defined times. Such approaches may be enforced by scripts which list all time events and associated actions. Time-driven models allow user to define the network and to make it evolve with time. As an example of such emulators, we can notice IREEL [18] and W-NINE [17]. Both emulators use an XML script containing update messages for a Dummynet static emulator used as an impairment engine. As an example, the emulated network can be designed to behave differently during the day and during the night.

Packet numbers in a flow can also be used as events, as in the KauNet network emulator [23]. In that case, the ipfw tool of FreeBSD is used to select a flow, and data-driven patterns are used to define how the impairments change with packet numbers. For instance, a packet loss pattern will use zeros for packet drops and ones for packet deliveries. KauNet also supports bit-error patterns, bandwidth change patterns, delay patterns and reordering patterns. Such patterns can also be used in a time-driven way, thus offering a more classical time-driven approach.

Other types of events have been proposed. Randomly generated events can be used to emulate random node failures. With an emulator able to read packet contents, a specific content (I image) or header value (DCCP handshake) can be detected and used as a triggering event. More generally, the metrology of the traffic can be used to detect specific conditions, such as the amount of flow reaching some level, to trigger specific impairments such as halving the bandwidth on the link.

Note that the most natural way to use the event-driven approach is the use of scripts associating impairment parameters with events, either explicitly like XML scripts IREEL or W-NINE, or implicitly like patterns and scenarios in KauNet.

5.1.3 Trace-based approach

This approach [31] is more realistic because the behavior of the network is obtained and will be reproduced exactly in the same way.

First, a collection phase is usually done by using probes. These probes are used to record the dates of packets arriving or leaving a host. The results are transmitted to a controller that evaluate the delay and the mean loss rate to give the basic network model. This allows the user to get dynamic network profile. The limitation of the trace-based approach is that it cannot reproduce all conditions a network would experience. A single trace can only capture a snapshot of the varying performance along a particular path. Furthermore, the traces cannot fully reproduce the network behavior because it is non deterministic. The same situation in another time could have produced different parameters.

The advantages of the trace-based approach is to use existing traces representing complex mobility movement to evaluate a prototype [1, 35].

5.2 Virtualization

Network and system virtualization allow to easily manage multiple networks and systems, each of them customized to a specific purpose at the same time over the same shared infrastructure [38]. Nowadays, virtualization is perceived as the best candidate to support multiple router software candidate releases simultaneously as a long-run testing method before real deployment in the Internet (see for instance, Ref. [28]). However, in this paper we are more interested in the second role of virtualization which is to run simultaneously multiple experiments in a shared experimental facility.

The virtual architecture models are higher level models allowing the representation of a target network that is going to be emulated. It consists of two different aspects namely System Emulation and Real-time Discrete event simulation. This allows the design of an emulation model according to a real network topology where the experimented flow crosses a set of real or virtual nodes. Another way to achieve this is to use the Real-time Discrete Event Simulation through the establishment of a bridge between real packets and simulated event-driven environments as in the ns-2 emulation extension NSE [21].

The global network behavior is produced by virtually reproducing the network topology and components.

Two directions are taken depending on the way this virtualization is achieved. In the virtual systems, all nodes constituting the target network to be emulated are implemented either onto a single centralized system (several virtual nodes co-exist into the centralized system) or distributed onto various distinct systems (e.g., a computing grid) usually connected together by high speed networks. Virtual links are used to connect these nodes together according to the topology of the targeted network. Real protocols such as IP or routing ones can also be implemented into the virtual node system. Of course, in this type of architecture, the classical strategy to produce realistic behavior is to introduce real traffic into the emulated network to produce congestion, delays, losses, etc.

IMUNES [39] is an example of centralized virtual node approach. It proposes a methodology for emulating computer networks by using a general purpose OS kernel partitioned into multiple lightweight virtual nodes. The virtual nodes can be connected via kernel level links to arbitrarily form complex network topologies. Furthermore, IMUNES allows to emulate fully functional IP routers over each emulated virtual nodes. IMUNES provides each virtual node with an independent network stack, thus enabling highly realistic and detailed emulation of network routers. It also enables user-level applications to run within the virtual nodes. At user level, IMUNES proposes a very convenient interface allowing to easily define the emulated network, namely the virtual nodes, the software, the links, and the impairment parameters.

The Entrapid protocol development environment [27] introduced a model of multiple virtualized networking kernels, which presents several variants of the standard BSD network stack in multiple instances, running as threads in specialized user process. Other

Table 1 Summary of the emulators cited in this paper

Name	Hardware model —comments	Impairments model	Network model	Year	Web
Dummynet [34]	Centralized —FreeBSD, IP-level emulation	Kernel	Static	1997	[2]
KauNet [22]	Centralized —based on Dummynet	Kernel	Event driven	2006	[7]
IREEL [18]	Distributed —based on Dummynet	Kernel	Event driven	2006	[6]
Netem [25]	Centralized —GNU/Linux, IP-level emulation	Kernel	Static	2005	[8]
NIST Net [16]	Centralized —sophisticated statistical distributions	Kernel	Static	2003	[9]
ONE [11]	Centralized —the first network emulator	User	Static	2001	[11]
PlanetLab [32]	Distributed —based on Linux virtual machines	Kernel	Virtualization	2003	[13]
EmuLab [36]	Distributed —based on Dummynet links	Kernel	Virtualization	2001	[3]
IMUNES [40]	Centralized —based on FreeBSD virtual machines	Kernel	Virtualization	2003	[5]
Alpine [20]	Centralized	Kernel	Virtualization	2001	No
Entrapid [27]	Centralized	Kernel	Virtualization	1999	No
Virtual routers [15]	Centralized	Kernel	Virtualization	2003	No
GNS3 [4]	Centralized —based on Cisco router images	Kernel	Virtualization	2008	[4]
NETShaper [26]	Distributed —link-level emulation	Kernel	Event driven	2002	No
Orbit [33]	Distributed —two-tier laboratory emulator/field trial network testbed	Kernel	Event driven	2005	[12]
W-NINE [17]	Distributed —wireless emulator	Kernel	Event driven	2008	No
CMU [31]	Centralized —seminal paper on trace-based approach	User	Trace based	1997	No

approaches that follow this approach is the Alpine emulator [20] project, GNS3/dynamips [4], and Virtual Routers [15].

The virtual architectural approach is often considered as the only mean to achieve realistic emulation of complex network topology. As previously introduced, PlanetLab proposes to directly use the Internet links to obtain real measures (for metrology purpose) conjointly with an emulation system allowing to map several end-hosts on a single computer in order to drive several and different experiments in parallel. Nevertheless, the major problem of this approach may be the scalability issue. Issues such as how to implement one or several core network routers in a single machine and how to manage the number of flows in a centralized manner remain problematic. These questions are difficult to answer, not only in the context of total centralization but also in the context of distributed systems such as grids.

6 Summary

We propose in this section a summary of the main characteristics of the emulators cited in this paper (following our classification model presented in Table 1). Note that the trace-based approach is a functionality that is already included inside some emulators (Orbit, W-NINE, KauNet, etc.) and can be added as a preprocessing tool to any event-driven emulator. However, we do not list this capability when it corresponds to an option and is not used as default network model.

7 Conclusions

This paper attempts to provide highlights concerning network emulation which is considered to be in the middle between simulation and live-testing schemes. We saw that network emulation combines the advantages offered by simulation and live testing at the same time while allowing different evaluation metrics (i.e., processing overhead, memory footprint). Another important finding is that we can easily set up complex measurements testbed by combining both virtualization and network emulation tools. However, network emulation is definitely not the unique answer and must be carefully weighted as a function of the performances an experimenter seeks to evaluate. Thus, to obtain a clear view, we develop an architectural model which illustrates and classifies all types of emulation tools. We hope both model and arguments presented would help the reader to better weight emulation in an evaluation process and choose the right scheme to assess the performances targeted.

References

1. CRAWDAD (2011) Community resource for archiving wireless data At Dartmouth. http://crawdad.cs.dartmouth.edu/
2. Dummynet (1997) http://info.iet.unipi.it/~luigi/dummynet/
3. Emulab (2001) http://www.emulab.net/
4. GNS3 (2008) Graphical network simulator. http://www.gns3.net
5. IMUNES (2003) http://imunes.tel.fer.hr/imunes/
6. IREEL (2006) http://ireel.npc.nicta.com.au/
7. KauNet (2006) http://www.kau.se/en/kaunet
8. Netem (2005) http://www.linuxfoundation.org/collaborate/workgroups/networking/netem"
9. Nistnet (2003) http://snad.ncsl.nist.gov/nistnet/
10. ns-2 user manual (2011) http://www.isi.edu/nsnam/ns/
11. One user manual (2001) http://irg.cs.ohiou.edu/one/manual.html
12. Orbit (2005) http://www.orbit-lab.org/
13. Planetlab (2003) http://www.planet-lab.org/
14. Opnet technologies (1991) http://www.opnet.com
15. Baumgartner F, Braun T, Kurt E, Weyland A (2003) Virtual routers: a tool for networking research and education. ACM SIGCOMM Comput Commun Rev 33(3):127–135
16. Carson M, Santay D (2003) NIST Net: a linux-based network emulation tool. ACM Comput Commun Rev 33(3):111–126
17. Conchon E, Pérennou T, Garcia J, Diaz M (2010) W-NINE: a two-stage emulation platform for mobile and wireless systems. EURASIP Journal on Wireless Communications and Networking 2010, Article 149075, 20 pp. See for details: http://www.hindawi.com/journals/wcn/2010/149075/cta/
18. Dairaine L, Jourjon G, Lochin E, Ardon S (2007) Ireel: remote experimentation with real protocols and applications over an emulated network. ACM SIGCSE Bull Inroads 39(2):92–96
19. Dawson S, Jahanian F (1995) Probing and fault injection of distributed protocols implementations. In: International conference on distributed computer systems
20. Ely D, Savage S, Wetherall D (2001) Alpine: a user-level infrastructure for network protocol development. In: USITS'01 Proceedings of the 3rd conference on USENIX symposium on internet technologies and systems
21. Kevin Fall (1999) Network emulation in the VINT/NS simulator. In: IEEE fourth symposium on computers and communications
22. Garcia J, Alfredsson S, Brunstrom A (2006) The impact of loss generation on emulation-based protocol evaluation. In: PDCN'06: proceedings of the 24th IASTED international conference on Parallel and distributed computing and networks. Anaheim, CA, USA
23. Garcia J, Conchon E, Pérennou T, Brunstrom A (2007) Kaunet: improving reproducibility for wireless and mobile research. In: MobiEval System evaluation for mobile platforms, workshop of mobisys 2007, San Juan, Puerto Rico, pp 21–26
24. Handley M, Floyd S, Pahdye J, Widmer J (2003) TCP-Friendly Rate Control (TFRC): protocol specification. Request For Comments 3448, IETF
25. Hemminger S (2005) Network emulation with netem. In: Australia's national linux conference (LCA) Canberra, Australia

26. Herrscher D, Rothermel K (2002) A dynamic network scenario emulation tool. In: 11th International conference on computer communications and networks
27. Huang XW, Sharma R, Keshav S (1999) The ENTRAPID protocol development environment In: IEEE Infocom
28. Keller E, Yu M, Caesar M, Rexford J (2009) Virtually eliminating router bugs. In: ACM CoNext
29. Siracusa R, Ott M, Seskar I, Singh M (2005) Orbit testbed software architecture: supporting experiments as a service. In: IEEE Tridentcom 2005, Trento, Italy
30. Mills D (1992) Network time protocol (version 3) specification, implementation. Request For Comments 1305, IETF
31. Noble BD, Satyanarayanan M, Nguyen GT, Katz RH (1997) Trace-based mobile network emulation. In: ACM SIGCOMM Cannes, France
32. Peterson L, Culler D, Anderson T, Roscoe T (2002) A blueprint for introducing disruptive technology into the internet. In: 1st Workshop on hot topics in networks (HotNets-I) Princeton, New Jersey, USA
33. Ramachandran K, Kaul S, Mathur S, Gruteser M, Seskar I (2005) Towards large-scale mobile network emulation through spatial switching on a wireless grid. In: Workshop on experimental approaches to wireless network design and analysis, (E-Wind), ACM Sigcomm
34. Rizzo L (1997) Dummynet: a simple approach to the evaluation of network protocols. ACM Comput Commun Rev 27(1):31–41
35. Scott J, Hui P, Crowcroft J, Diot C (2006) Haggle: A networking architecture designed around mobile users. In: The third annual IFIP conference on wireless on-demand network systems and services (WONS 2006), Les Menuires, France
36. White B, Lepreau J, Stoller L, Ricci R, Guruprasad S, Newbold M, Hibler M, Barb C, Joglekar A (2002) An integrated experimental environment for distributed systems and networks. In: Fifth symposium on operating systems design and implementation, pp 255–270
37. Widmer J (2000) TFRC userspace prototype. http://aciri.org/tfrc/code/
38. Yu M, Yi Y, Rexford J, Chiang M (2008) Rethinking virtual network embedding: substrate support for path splitting and migration. ACM SIGCOMM Comput Commun Rev 38(2):17–29
39. Zec M, Mikuc M (2004) Operating system support for integrated network emulation in IMUNES. In: First workshop on operatings System and architectural support for the on demand IT infraStructure, Boston, USA
40. Zec M, Mikuc M (2004) Operating system support for integrated network emulation in imunes. In: 1st Workshop on operating system and architectural support for the on demand IT infraStructure/ASPLOS-XI, Boston, USA
41. Zheng P, Ni LM (2003) EMPOWER: a network emulator for wireline and wireless networks. In: IEEE Infocom San Francisco