The Effect of Hardware/Software Features on the Performance of an Open–Source Network Emulator

Domenico Capriglione[1], Gianni Cerro[2(✉)], Luigi Ferrigno[2], and Gianfranco Miele[2]

[1] University of Salerno, Fisciano, Italy
dcapriglione@unisa.it
[2] University of Cassino and Southern Lazio, Cassino, Italy
{gianni.cerro,luigi.ferrigno,g.miele}@unicas.it

Abstract. The authors investigate a network emulator performance versus the variability of hardware/software features of the hosting machine. In particular, the evaluation of static and dynamic delays is carried out considering several testing conditions. In detail, as concerns emulator configurations, the influence of packet rates on imposed delays values and distributions are analyzed; as for hardware and software, different values for RAM, CPU cores and operating system are tested. Results, reported as mean values and standard deviation, show two main trends: the resource availability has an important impact on the emulation stability and on the measurement repeatability; secondly, higher differences in performance levels for low imposed delay values, which is the most interesting zone in a few milliseconds latency world. The paper aims to show that the capability to emulate network impairments is generally influenced by hardware/software capabilities and it must be considered when using network emulation for specific test purposes.

Keywords: Computer networks · Network emulation · QoS measurements · System virtualization

1 Introduction

Modern testing schemes, in most engineering fields, are especially focused on simulations, model planning, test execution in a software–based environment and extrapolation of theories and laws on the basis of the obtained results. This is particularly true in telecommunication and computer science [16]. Protocol and novel communication technology tests usually rest on simulators [14], and emulators [9] having the purpose to replicate the real environment where commercial releases should find their place. In this field, an important role is played by network emulation [7,13,18]. It has the goal to emulate all possible impairments a telecommunication network can experience in real cases, such as delays,

Published by Springer Nature Switzerland AG 2019
M. Di Felice et al. (Eds.): WWIC 2019, LNCS 11618, pp. 233–245, 2019.
https://doi.org/10.1007/978-3-030-30523-9_19

jitters, packet losses, packet duplication, just to cite a few. Emulators can be hardware [15] or software [1,3]. Usually hardware emulators are costly but they present certified level of performance, while software emulators and, particularly, open source software emulators are free, flexible but their reliability is left as an open issue to be investigated.

Starting from previous experience of the authors in such field [2,5], this paper describes the investigation of the effect of hardware/software available resources on the performance of a widely–known open source network emulator, namely Net–Em [10,11], that is part of the Linux traffic control facilities and allows to emulate several impairments in a network link. To achieve such goal, there are two main possibilities: having several physical machines with different hardware or applying a virtualization process and changing resource availability via software. In this paper, the second choice has been selected. Virtualizing the system leads to other sources of uncertainty and several papers address the issue [8,12,17]. In our case, those contributions have been deliberately neglected, since the aim of the work is to evaluate the effect of having limited–hardware/heavy–software resources on the obtained performance. All results are presented in relative values, taking as nominal performance the one experienced with the maximum available resources. The output of this work represents a step forward in the way to assign a metrological value to a software network emulator that, through a self-calibration procedure, could provide the final user with a quantitative evaluation of its own expected capabilities in reproducing the required impairments.

The organization of the paper is the following: Sect. 2 is going to describe the adopted methodology along with the planned test conditions; Sect. 3 is intended to present obtained results both in terms of mean value and standard deviations. Section 4 discusses about the meaning of the obtained results in terms of general statements the paper wants to convey and it also provides possible improvements and developments of the experimental set–up.

2 Materials and Methods

In this section, we provide a detailed description of the adopted test set–up. It consists of the adopted network emulator, namely Net–Em, the virtualization schema, how the measurements are performed and which test conditions represent the most interesting scenarios to be investigated.

2.1 The Adopted Network Emulator

Net–Em (abbreviation for Network Emulator) can be seen as a tool inside traffic control (TC) routines by Linux Systems [10,11]. For a given network, composed of several links, its functionalities allow to add impairments such delays, delay variations (jitter), packet losses and other limitations to one or more links of the networks. In this paper, only delay is considered as parameter of investigation.

As delay setting regards, Net–Em requires three input parameters: the mean value (μ), the standard deviation (σ) and the correlation coefficient (ρ). If not differently specified, it generates a uniform distribution using as parameters μ and σ and considering ρ to correlate currently generated delay with the previous one (associated to the previous packet). Furthermore, thanks to *iproute2* tool collection, other random distributions can be associated to delay generation, such as normal, Pareto and Pareto normal functions.

2.2 The Virtualized System

The realized test set–up is composed of three virtual machines residing on the same PC. Figure 1 provides a sketch of the realized system. It is based on VirtualBox, that is a virtualization product provided by Oracle, able to work on all main operating systems and it is available for free as Open Source Software under GPL terms. Main functionalities are summarized by Oracle in their white paper [6].

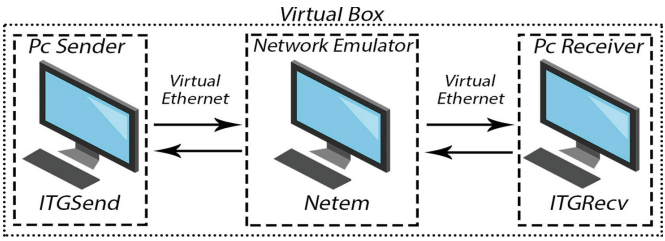

Fig. 1. A sketch of the virtualized system

Inside VirtualBox container, to create a network topology, three different virtual machines have been installed with the following tasks:

- PC Sender: this machine is responsible to send data, with different features in terms of traffic duration, packet rates and contents.
- Network Emulator: it is the core of the system. It receives data from the sender and forwards them to the PC Receiver. According to imposed test conditions, the forwarding process is consequently perturbed.
- PC Receiver: this machine is responsible to receive data and store them for further processing.

As reported in Fig. 1 (see *ITGSend* and *ITGRecv*), a suitable software is adopted to generate and retrieve traffic data [4].

An important tool of this suite is *ITGDec*, that can be used off–line to analyze traffic features, as packet delays, packet-losses etc.

To be able to use Net–Em, all virtual machines are equipped with Linux Operating Systems: Ubuntu 17.10 on PC Sender and PC Receiver, while different versions on Network Emulator in order to assess the influence of software, as well.

In terms of hardware equipment, both PC sender and receiver are given with 2 GB RAM and dual–core processors.

2.3 The Measurement Procedure

The authors have developed a measurement procedure proposed in [5]. This procedure is adopted in current paper as well, to verify performance levels of the proposed set–up and emulator.

In particular, measurement procedure is reported in Fig. 2, where the timeline of the test operations is depicted.

It consists of three main steps:

- initialization;
- machine time synchronization;
- Traffic flow activation and data storage.

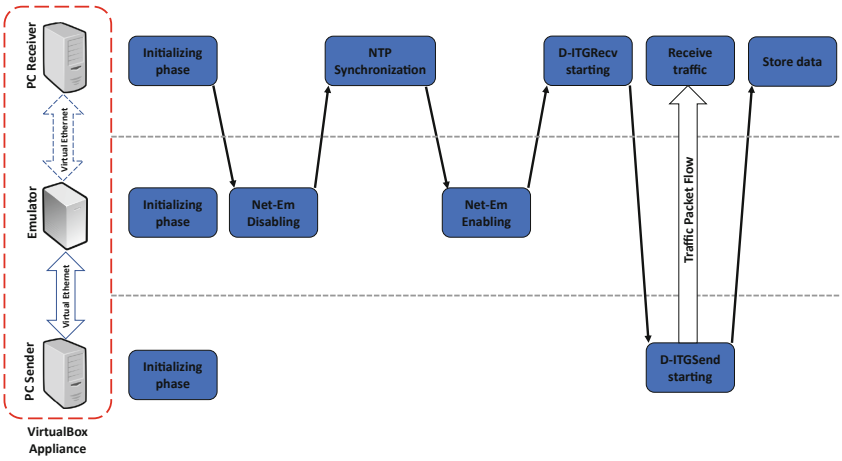

Fig. 2. The adopted measurement procedure

In detail, during initializing phase, traffic profiles, emulator settings and receiver logging function are adjusted. In the second step, machine time synchronization is carried out. In particular, to avoid any influence, the emulator machine works only as a relay machine able to forward traffic from PC Sender to PC Receiver without adding any impairment condition. In this case, adopting Network Time Protocol (NTP), we aligned Sender to Receiver absolute time. In this way, delay estimation is coherently computed, adopting as sending instant the time-stamp included by the Sender in each packet header.

The third step consists of enabling the emulator, setting suitable parameters, such as delay value and distribution and link of interest for impairment application. After setting–up testing conditions, receiver and sender are respectively

activated and traffic flow is started. Packets pass through the emulator, which corrupts the link to the receiver. They are finally received by PC receiver and there stored to be off–line processed.

2.4 Performed Test Conditions

The evaluation of performance is subjected to the imposition of several test conditions.

In particular, we can divide test conditions in two categories:

- imposing delay conditions to the emulator;
- changing hardware/software features of the network emulator hosting machine.

As regards imposed delay conditions, we set:

- static delays: $[0, 5, 20]$ ms;
- dynamic Gaussian delays: mean value $\mu = 20$ ms and standard deviations $\sigma = [1, 2, 5]$ ms.

As regards hardware/software features, we set:

- CPU cores: $1, 2, 4$;
- RAM memory: $[500, 2048, 4096]$ MB;
- Operating System: Ubuntu Desktop, Ubuntu Server.

Furthermore, we tested each delay value under several network stressing conditions, i.e. by imposing different packet rates to the data flow. In detail, adopted packet rates are: [200 500 1000 2000 5000 10000] pkt/s. The data flow duration is fixed to 15 s.

For each operating condition (combination of Net–Em and hardware features), 20 tests are performed to estimate repeatability of the obtained measurements.

3 Results

Measurement campaign results are reported in terms of mean value and standard deviation. Their representation is organized in:

- effect of CPU core number variation on static (Figs. 3 and 4) and dynamic delays;
- effect of RAM variation on static (Figs. 5 and 6) and dynamic delays (Table 1);
- effect of Operating System variation on static (Figs. 7 and 8) and dynamic delays (Table 2).

3.1 Static Delay Evaluation

CPU Effect. As first test group, static delays have been imposed to Net–Em. Three different values are considered: 0 ms, 5 ms and 20 ms and results are evaluated by testing the emulator under different CPU core numbers. In particular we consider 1, 2 and 4 cores. To be cautionary, 4 GB RAM have been used and Ubuntu Server Environment has been adopted as operating system version.

Most critical situations are those referred to lowest imposed delays, namely 0 ms and 5 ms. In 0 ms case, the emulator is working only as a relay machine, without adding any delay on the path. The experienced delay is always greater than zero, due to the virtual link intrinsic delay. Such value is decreasing with packet rates and it does not exhibit a clear trend behavior with respect to core variation. The only clear difference is in terms of standard deviations: in detail, 2 core situation, reported in orange in Fig. 3, exhibits the most repeatable behavior for any specific test condition (this phenomenon is reported with the green vertical bar in the same figure). Generally, 4–core–configuration outperforms single core case, both in terms of lower average value and standard deviation. Therefore, for 0 ms case, the best configuration is 2–core and the worst one is single core.

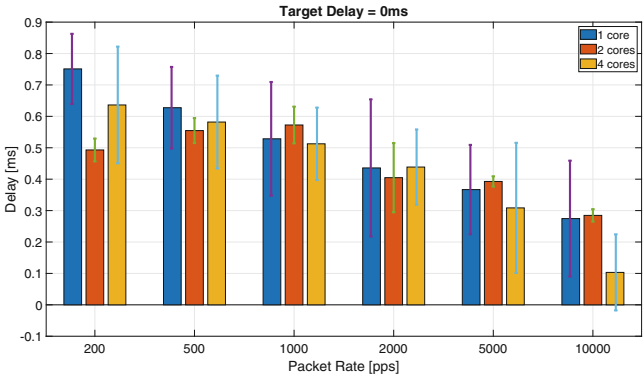

Fig. 3. Static Delay results with target value equal to 0 ms. Effect of CPU core number variation.

In 5 ms case (see Fig. 4), delay mean values have a lower variability. In particular, there is a light overestimation for all packet rate conditions, and core number influence is less evident. Still, 2–core–case has a very stable behavior and a very low standard deviation. Even if not reported for a sake of brevity, when imposed delay is equal to 20ms, slight differences among different hardware features are negligible, and results are in–line with target performance, except at 200 packet rates, where single core case has a lower repeatability.

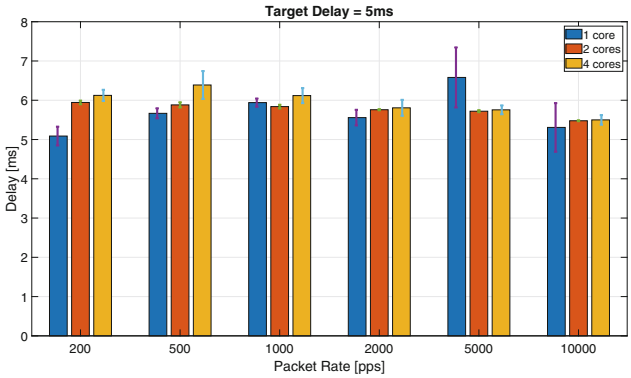

Fig. 4. Static Delay results with target value equal to 5 ms. Effect of CPU core number variation.

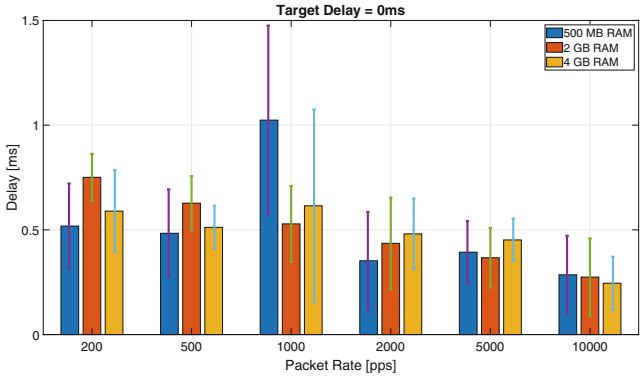

Fig. 5. Static Delay results with target value equal to 0 ms. Effect of RAM size variation.

RAM Effect. To evaluate the central memory effect on the performance of the network delay emulation, we tested three different and typical RAM sizes: 500 MB, 2048 MB, 4096 MB. As core evaluation on static delays has proved the suitability of using 2–core case, that is the CPU configuration adopted for RAM tests. Also in this case we use 0 ms, 5 ms and 20 ms as target delay values. When we consider 0 ms case, i.e. Fig. 5, values are generally higher than the ones obtained when core influence is evaluated. In particular, very high standard deviations have been obtained, especially for 500 MB case. In this figure, result trends can be divided into two cases: before 1000 pkt/s and after 2000 pkt/s. In the first part, 2 GB case appears as the best configuration in terms of repeatability, while in the second part 4 GB results exhibit a lower standard deviation.

This is an expected behavior, since when packet rates are higher, the number of packets and the data flow size increase and wider memory availability can

help in managing emulation. Data size is, in any case, smaller than available memory. As a general statement, measurements result in any case compatible, where the concept of compatibility in measurement has been widely explained in [5]. Results obtained for 5 and 20 ms follow the trends already explained in case of core number variation. The best configuration in these cases results 4 GB RAM. Only 20 ms figure (see Fig. 6) is actually reported in the paper.

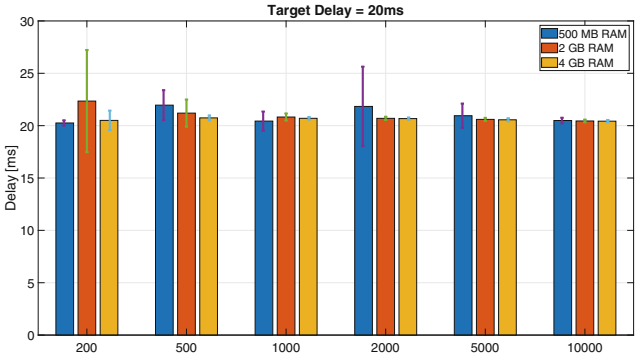

Fig. 6. Static Delay results with target value equal to 20 ms. Effect of RAM size number variation.

OS Effect. As for the effect of the Operating System on static delay emulation performance, we tested two different Linux Ubuntu versions, in particular Ubuntu Desktop 18.10 and Ubuntu Server 18.10. The idea is to use the same version of the operating system, in order to understand if the Graphical user interface and the processes connected to it could have an effect on the delay

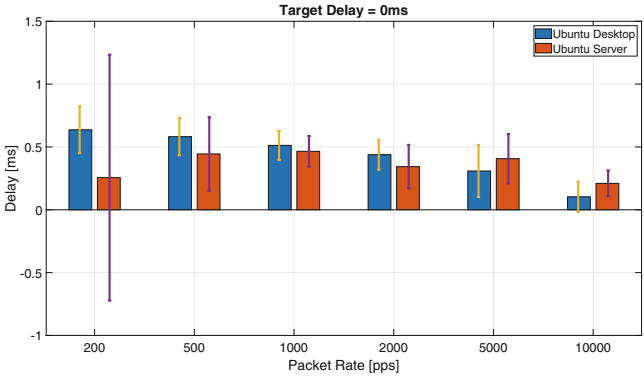

Fig. 7. Static Delay results with target value equal to 0 ms. Effect of Operating System variation.

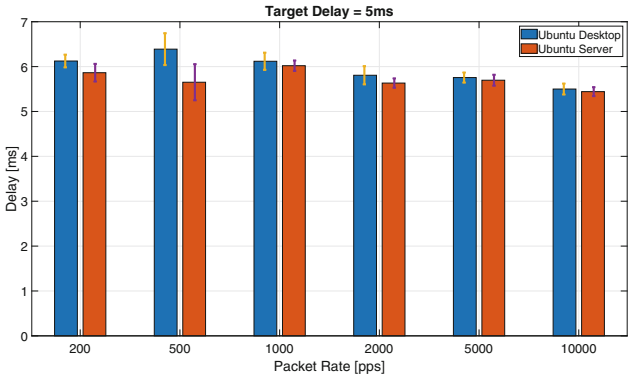

Fig. 8. Static Delay results with target value equal to 5 ms. Effect of Operating System variation.

emulation. Ubuntu Server is, indeed, a command–line operating system. Also in this case, we tested three static delays and results prove a generalized increase of the measured delay when Ubuntu Desktop is adopted. This phenomenon can be observed at lower packet rates for 0 ms delay (see Fig. 7), while it is always true when imposed delay values are set to 5 ms and 20 ms. Standard deviations have comparable values in all cases, except for a particular case at 200 pkt/s for 0 ms delay. In this case, Ubuntu Server case exhibits a really unstable behavior. For sake of brevity, only 0 ms and 5 ms delay cases are reported in Figs. 7 and 8.

3.2 Dynamic Delay Evaluation

The evaluation of Dynamic Delay emulation performance requires several levels of verification. Firstly, we tested only Gaussian Distribution. Therefore, when measurements are carried out, results must be evaluated in terms of average value, standard deviation and correspondence between the imposed probability density function (pdf) and obtained empirical pdf. In this subsection, we show results that are relative to all cited parameters.

In order to be concise, we present results in form of tables. In detail, for each influence factor (processor, memory, system), two tables report results obtained in two packet rate conditions: low traffic (200 pkt/s) and high traffic (5000 pkt/s). Since results in terms of mean and standard deviation values are very similar, we do not report all tables for sake of brevity. In particular, RAM and OS effects are reported only.

Reported tables prove how standard deviation values are often lower than the expected ones, except one case: Table 2, in 1 ms line, where all configurations (Desktop, Server) exhibits a higher value. This phenomenon can be referred to the limited capability of the emulator to reproduce distribution. Still, hardware resources are not responsible for these synthetic data, since they are all comparable.

Table 1. Packet Rate 200 pkts - Dynamic Delay - RAM effect

Imp. values (μ, σ) [ms, ms]	Obt. mean (512 MB) [ms]	Obt. mean (2 GB) [ms]	Obt. mean (4 GB) [ms]	Obt. std (512 MB) [ms]	Obt. std (2 GB) [ms]	Obt. std (4 GB) [ms]
(20,1)	21.05	20.91	20.99	0.99	0.95	1.00
(20,2)	21.07	20.91	20.98	1.71	1.70	1.70
(20,5)	21.09	20.97	21.01	4.22	4.20	4.24

Table 2. Packet Rate 200 pkts - Dynamic Delay - OS effect

Imp. values (μ, σ) [ms, ms]	Obt. mean (Desktop) [ms]	Obt. mean (Server) [ms]	Obt. std (Desktop) [ms]	Obt. std (Server) [ms]
(20,1)	21.02	22.60	1.03	1.20
(20,2)	21.03	22.57	1.72	1.88
(20,5)	21.05	22.48	4.25	4.34

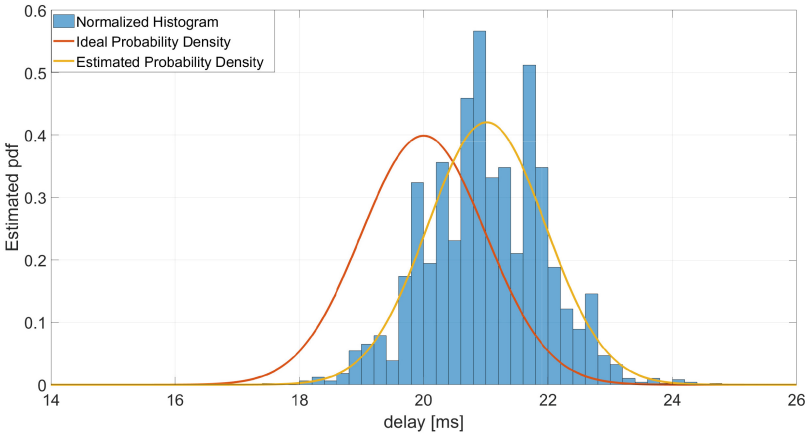

Fig. 9. Comparison of probability distributions: 500 MB RAM – 1 core CPU – Ubuntu Desktop OS (Color figure online)

Furthermore, in order to show adherence between observed and expected probability distributions, Figs. 9 and 10 are reported. Each figure is characterized by three information graphics:

- a red line: the ideal probability density function, as imposed in Net–Em settings;
- a yellow line: the derived probability density function, applying Gaussian analytic law to obtained mean and standard deviation;
- a normalized histogram: real data distribution.

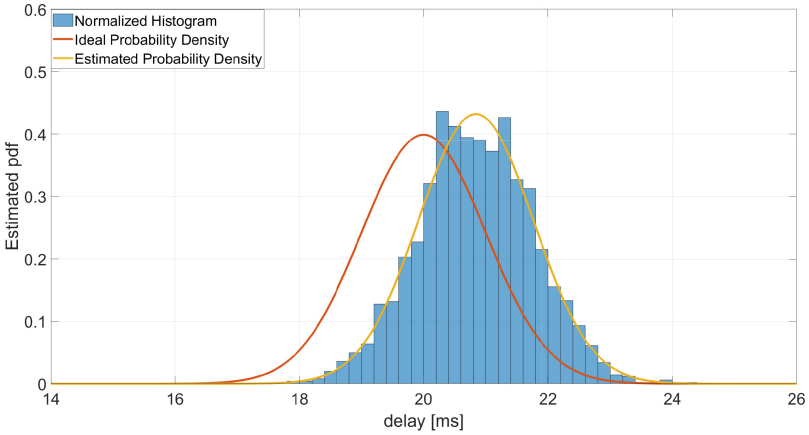

Fig. 10. Comparison of probability distributions: 2 GB RAM – 2 core CPU – Ubuntu Desktop OS (Color figure online)

What really makes the difference when comparing results obtained with different hardware resources are the actual probability density functions (briefly reported as distribution in the text, with some ambiguity). Indeed, when 500 MB RAM and 1 core are used, the obtained empirical distribution is quite far from being Gaussian, as the histogram bins do not appear to follow the yellow distribution line. When resources increase (see Fig. 10), experimental distribution approaches the expected one.

In both cases, the red line, i.e. the ideal distribution, is not well achieved by the data. This can be a problem related to the emulator itself, since it is not dependent on the adopted hardware resources.

4 Discussion

In this paper, an analysis of the impact of variable hardware resources on the performance of a common adopted network emulator is reported. As stated in the introduction, we did not mean to evaluate the emulator capabilities but their stability when different resources are available. We can affirm two main results: in static delay case, a sensitive impact can be observed in case of very low imposed delay (in our case, 0 ms) and with high imposed packet rates. When delay constraints are relaxed (5 ms or 20 ms), hardware resources become less important; in dynamic case, the emulator is capable to pretty adhere to synthetic values of a probability distribution, but hardware limitations appear as preeminent when the probability density function is analyzed. These results make the emulation possible also on low power machines, if some strict constraints can be relaxed and required high performance devices when precise and very low delays are desired. Further analyses will take care of other quantities, such as packet loss emulation, in variable resources scenarios.

References

1. Ahrenholz, J., Danilov, C., Henderson, T.R., Kim, J.H.: Core: a real-time network emulator. In: 2008 IEEE MILCOM, pp. 1–7, November 2008
2. Angrisani, L., Capriglione, D., Cerro, G., Ferrigno, L., Miele, G.: Experimental analysis of software network emulators in packet delay emulation. In: 2017 IEEE International Workshop on Measurement and Networking (M&N), pp. 1–6, September 2017. https://doi.org/10.1109/IWMN.2017.8078382
3. Beshay, J.D., Francini, A., Prakash, R.: On the fidelity of single-machine network emulation in Linux. In: 2015 IEEE 23rd International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 19–22, October 2015. https://doi.org/10.1109/MASCOTS.2015.18
4. Botta, A., Dainotti, A., Pescapè, A.: A tool for the generation of realistic network workload for emerging networking scenarios. Comput. Networks **56**(15), 3531–3547 (2012)
5. Capriglione, D., Cerro, G., Ferrigno, L., Miele, G.: How to quantify *trust* in your network emulator? In: Chowdhury, K.R., Di Felice, M., Matta, I., Sheng, B. (eds.) WWIC 2018. LNCS, vol. 10866, pp. 171–182. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-02931-9_14
6. Coter, S., King, G.: Oracle VM 3: Building a Demo Environment using Oracle VM VirtualBox. Technical report, Oracle Corporation (04 2016)
7. Deng, B., Wang, X., Jiang, M., Liu, Y.: An emulation architecture for the integration of virtual and physical networks. In: 2017 8th IEEE ICSESS, pp. 399–405, November 2017
8. Edwards Sr, T.S.: Systems and methods for improving virtual machine performance, US Patent 8,332,571, December 2012
9. Ferenc, G.Z., Dinic, M.D., Markovic, A.I., Jovanovic, P.D.: UHI boot protocol implementation in android emulator for MIPS architecture. In: 2017 25th Telecommunication Forum (TELFOR), pp. 1–4, November 2017
10. Hemminger, S., et al.: Network emulation with NetEM. In: Linux Conf Au, pp. 18–23 (2005)
11. Jurgelionis, A., et al.: An empirical study of NetEm network emulation functionalities. In: 2011 Proceedings of ICCCN, pp. 1–6, July 2011
12. Kousiouris, G., Cucinotta, T., Varvarigou, T.: The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks. J. Syst. Softw. **84**(8), 1270–1291 (2011)
13. Kretsis, A., Corazza, L., Christodoulopoulos, K., Kokkinos, P., Varvarigos, E.: An emulation environment for SDN enabled flexible IP/optical networks. In: 2016 18th ICTON, pp. 1–4, July 2016
14. Masruroh, S.U., Fiade, A., Iman, M.F., Amelia: Performance evaluation of routing protocol RIPv2, OSPF, EIGRP with BGP. In: 2017 International Conference on Innovative and Creative Information Technology (ICITech), pp. 1–7, November 2017
15. Nakauchi, K., Kobayashi, K.: Studying congestion control with explicit router feedback using hardware-based network emulator. In: Proceedings of PFLDNET 2005, Lyon, France (2005)
16. Sarkar, N.I., Halim, S.A.: A review of simulation of telecommunication networks: simulators, classification, comparison, methodologies, and recommendations. J. Sel. Areas Telecommun. (JSAT), 10–17 (2011)

17. Tickoo, O., Iyer, R., Illikkal, R., Newell, D.: Modeling virtual machine performance: challenges and approaches. ACM SIGMETRICS Perform. Eval. Rev. **37**(3), 55–60 (2010)
18. Zheng, P., Ni, L.M.: EMPOWER: a cluster architecture supporting network emulation. IEEE Trans. Parallel Distrib. Syst. **15**(7), 617–629 (2004). https://doi.org/10.1109/TPDS.2004.21