77 Evaluation of Lightweight and Distributed Emulation Solutions for Network Experimentation

Emerson Rogério Alves Barea, Cesar Augusto Cavalheiro Marcondes, Lourenço Alves Pereira Jr., Hermes Senger, and Diego Frazatto Pedroso

Abstract

Network emulation is an intermediate solution for supporting experimentation on new protocols and services which falls between the high fidelity of fully implemented networks and running simulation models executed. Lightweight emulation environments emulate entire networks on a single machine, thus enabling experiments that are much realistic and easy to use, at a fraction of cost and complexity when compared to real system. Scalability of a network emulation environment is very relevant when the experimentation scenario involves large amounts of networking devices, services, and protocols. In this paper we evaluate the scalability of some lightweight and distributed emulation environments. Experiments show the consumption of resources for each environment including memory, number of processes created, disk utilization, and the time required to instantiate models. Our analysis can be useful for experimenters to decide on which environment to use.

Keywords

Lightweight · Emulation · Experiment · Mininet · Scalability

E. R. A. Barea (✉)
IFTO, Palmas, Brazil
e-mail: emerson.barea@ifto.edu.br

C. A. C. Marcondes · L. Alves Pereira Jr.
ITA, São José dos Campos, Brazil
e-mail: cmarcondes@ita.br; ljr@ita.br

H. Senger · D. F. Pedroso
UFSCar, São Carlos, Brazil
e-mail: hermes@ufscar.br; diego.pedroso@ufscar.br

77.1 Introduction

Experimentation methods for study of network technologies usually involves emulation [1]. Network emulation meets the advantages given by physical networks, supporting use of real applications, without the higher cost and complexity of the physical environment, ensuring better accuracy than simulation [2–4].

Over time, new emulation techniques were employed. Nowadays, commonly used solutions support full emulation of complex network environments using lightweight container-based virtualization on a single host. However, although this type of emulation is widely used and its benefits are well known, there are scenarios where the computational requirements demanded from emulated network are superior to those supported by individual machine, whether the number of emulated nodes, maximum throughput desired, or other important parameter to experimenter [1].

This critical limitation has motivated the proposal of solutions that mix emulation and simulation to improve scalability [5,6]. However, pure container-based emulation solutions have also evolved and now supports distributed processing. Although they exist, these solutions have not yet had their scalability characteristics evaluated, making it difficult to define which technology meets specific scenarios, or which computational resources it requires for network experimentation.

In this work, we focus on evaluating the performance and scalability of Mininet distributed technology, a lightweight container-based emulation solution that support distributed processing and can be used in network experimentation. Among the highlights and most important aspects, this work (1) preliminarily analyzes technical implementation features of Mininet Cluster and MaxiNet to identify

© Springer Nature Switzerland AG 2020

S. Latifi (ed.), *17th International Conference on Information Technology–New Generations (ITNG 2020)*, Advances in Intelligent Systems and Computing 1134,
https://doi.org/10.1007/978-3-030-43020-7_77

which technology allows lowest expenditure of general computational resources; and (2) evaluates its behavior regarding the utilization of computational resources in scenarios with specific requirements, identifying benefits and disadvantages of each implementation.

The relevance of this work is justified by the importance of comparing technical aspects of lightweight and distributed virtualization solutions for network experimentation, generating data to be used for identify which solution is indicated in each demand, facilitating the recognition of requirements needed by different scenarios.

This paper is organized as follows: Sect. 77.2 presents related works and preliminary concepts that will be used as information base for development of this work. Section 77.4 details procedure and technologies used in distributed container-based emulation evaluation. Section 77.5 presents and commented performance tests results; and Sect. 77.5 concludes and presents future work suggestions.

77.2 Theoretical Grounding and Related Work

This section presents the background and related works used in this work.

77.2.1 Lightweight Virtualization

The rise of the virtualization approach is not recent. Both platform and software layer responsible for abstracting and sharing hardware with necessary isolation levels between virtualized functions evolved, starting from O.S. virtualized over hypervisors with high function overhead, to lightweight virtualization based on process isolation at the same userspace (containers) [7], using concept of namespaces to isolate functions, such file system, process tree, network, user area, and others; and Control Groups (CGroups) limiting resources such memory, CPU and throughput [8].

Due to its characteristics, container-based virtualization proved feasible as a base for network experimentation, mainly because it supports the emulation of whole environments that support the study of diverse and complex scenarios, with a small overall cost and good precision results [1]. Currently, there are several tools that supports this kind of virtualization, and some of them are well known and widely applied in the most varied environments, like LXC, LXD and Docker.

Besides full-featured containers solutions that implement a large set of isolation mechanisms, there are also tools developed focused specific on emulation for network experimentation, such Mininet. Mininet differentiates itself from other implementations mainly using only the network and mount point namespaces attached to a Linux process, conferring less overhead on each container. It also uses CGroups for resource control, manages SDN switches and controllers, interconnecting all network emulated elements with virtual interfaces (veth). It has a programmable API accessible by command-line, allowing administration and control of all elements in an experiment.

Even these solutions shares container-based virtualization concept, there are implementation differences that lead to divergent behavior regarding computational resources utilization. Figure 77.1 shows the amount of memory used, the number of Linux processes spawed, and disk space used in scenarios where was created only 1 and 100 containers, using Mininet, LXC, LXD and Docker solutions.

Analyzing system utilization shown in Fig. 77.1 it is possible to note that Mininet consumes about 90 MB of memory on physical host when instantiated 100 containers, while LXC consumes 260 MB, Docker 350 MB and LXD 1.02 GB of memory. LXC, LXD and Docker RAM utilization is associated to the characteristics of system image used to create containers, since it carries all set of codes, libraries, environment variables and configuration files needed for its

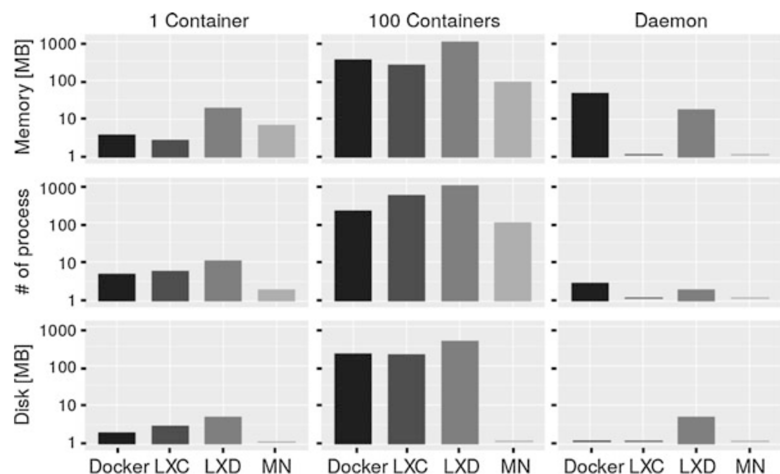

Fig. 77.1 Memory consumption, number of processes created and disk utilization in light virtualization solution

operation. For this reason, we used the smallest and simplest available images in official repositories, consuming as few resources as possible in container creation with each technology. For this, we used Linux Alpine 3.4 AMD64[1] image in LXD and Linux BusyBox 1.29 AMD64[2] in LXC and Docker containers.

Docker and LXD has a daemon responsible for manage all container operations, Docker daemon uses 58 MB of memory while LXD uses 18 MB. As for the amount of processes created for instantiation of 100 containers, Mininet creates 101 processes, Docker 202, LXC 501, and LXD 902 processes. Resources consumed for LXC, LXD and Docker is impacted again by characteristics of system image used.

For disk usage, Mininet makes no use since its containers uses same mount point common to all users on physical host file system, while LXC and LXD consume 210 MB and 430 MB of disk space each, for creation of independent virtual file system for each container. Docker, otherwise, consumes about 140 MB of disk space for 100 containers creation, because of copy-on-write technique.

Based on the data presented, it is possible to note that Mininet consumes less physical host resources for creating simple containers when compared to other technologies, therefore, Mininet can be considered enough for network experimentation.

77.2.2 Lightweight and Distributed Virtualization

When the concept of lightweight virtualization is extended to a distributed system, it is necessary to consider the procedures for container communication between cluster nodes and the techniques used for link bandwidth and delay parametrization.

Mininet supported distributed processing using SSH tunnels between containers on cluster nodes from version 2.2.0 (named Mininet Cluster) and GRE tunnels from version 2.3.0, supporting higher throughput because GRE does not use TCP for data transport. The creation of tunnels between cluster nodes is done automatically when a link is solicited between two elements in Mininet API and can connect any kind of elements. This feature is very important when a non-SDN network is emulated and only Mininet *host* element is used in an experiment, reducing the number of containers for not needing switches for remote element connection. Despite this advantage, Mininet Cluster not support link bandwidth and delay parametrization [9]. Mininet Cluster has pre-defined algorithms for elements placement automatically on cluster nodes, where *SwitchBinPlacer* distributes switches and controllers in blocks of uniform size based on cluster size, trying to allocate *host* elements on same node as switches; and *RandomPlacer* that does random elements distribution through cluster nodes [10].

Another distributed Mininet implementation is MaxiNet, which comprises an API acting as an administrative layer for Mininet, where a central node, called Frontend, invokes commands on remote nodes, called Workers, managing Mininet elements localized on cluster by a PYthon Remote Object (Pyro4) name server. MaxiNet uses GRE tunnels for remote element connection, however, these tunnels are only supported between *switch* element, an important limitation when compared to Mininet Cluster. MaxiNet supports link bandwidth and delay parametrization in its API and creates automatically GRE tunnels in cluster nodes. Uses METIS[3] library for topology graph partitioning, creating partitions with equivalent weights on all cluster nodes, aggregating most of traffic emulated in workers through minimum cut criterion based on topology links bandwidth [11].

Due to implementation differences between Mininet Cluster and MaxiNet, there are differences in behavior between then when looking at cluster computational resource consumption. However, for the best of our knowledge, there are no works that evaluate and compare their behavior regarding computational resource consumption when analyzing the scalability of these solutions.

77.3 Materials and Methods

In this section we describe the method used to evaluate Mininet distributed emulation solutions, detailing the procedure used and parameters adjustment for each technology.

77.3.1 Parameters and Measurement Procedure

The evaluation was based on behavior and resources consumption analysis in a cluster executing Mininet Cluster and MaxiNet technologies. To do this, we emulated datacenter networks topologies composed by Mininet's *host* element, representing network servers; *switch* element, representing switches used to interconnect servers; and *link* element, that connects servers and switches. In tested scenarios we variated the amount of each Mininet element created, the amount of cluster nodes allocated and distributed Mininet solution used.

Throughout the creation and activation process of Mininet elements, we monitored the effect of a single factor variation, or a set of them, in the amount of RAM used, the number of Linux processes spawned, remote network connections established between cluster nodes, and time spent per action of the distributed Mininet solution setup.

[1] https://alpinelinux.org/.

[2] https://busybox.net/.

[3] http://glaros.dtc.umn.edu/gkhome/metis/metis/overview.

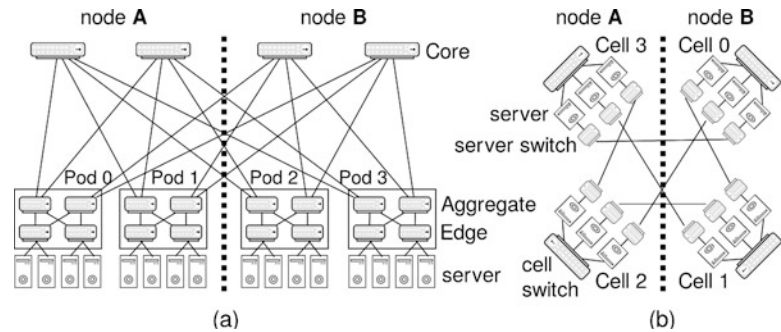

Fig. 77.2 (**a**) FatTree and (**b**) DCell topologies used in distributed Mininet solution evaluation

77.3.2 Topologies and Partitioning

We emulated two datacenter topologies with different computational requirements. First one is FatTree (Fig. 77.2a), which follows the structure of a k-naria tree, being recognized for supporting high throughput capacity between network nodes using generic switches. It consists of k groups of switches (pods), each containing two layers with $k/2$ switches each (Aggregate and Edge), with k ports per switch, and $(k/2)^2$ servers. Each Edge switch is connected to $k/2$ servers and $k/2$ Aggregate switches. Above pods there is another layer (Core) with $(k/2)^2$ switches, each Core switch is connected to k-pods via Aggregate switches [12]. As an example, a FatTree topology $k = 4$ has 4 pods with 4 servers, 2 Edge switches and 2 Aggregate switches each, plus 4 Core switches, totaling 16 servers, 20 switches and 144 links throughout the topology.

DCell topology (Fig. 77.2b) contains k cells composed by t servers each, where $t = k - 1$. The t servers of a cell are interconnected through independent switches to each cell, as well each server$_{k|t}$ is connected directly to another server$_{t+1|k}$ through a link [13]. As an example, a DCell topology $k = 4$ has 4 cells with 3 servers and 1 switch each, totaling 4 switches, 12 servers and 18 links in topology.

In order to ensure equal distribution of processing load in cluster nodes, we used *Round Robin* technique to partition the k elements of each topology between cluster nodes, also allocating all Mininet *host* elements of a single pod, in FatTree topology, and of a single cell, in DCell topology, on the same cluster node.

Because MaxiNet supports remote connection only between *switch* type Mininet elements, as shown in Sect. 77.2.2, we included switches between the links that connect servers in DCell topology. We identified these switches as $serverswitch$ in Fig. 77.2b.

77.3.3 Test Methodology

Distributed Mininet solutions evaluation depends of control factors parameters adjustment, which are tested simultaneously and can assume different levels. For each combination of possible levels, it is necessary to execute one or more test sequences that generate results to be analyzed, but the setting of the adjustment in each parameter is not trivial, and the increase in the number of factors and tested levels increases considerably the size of the experiment, and may even make it impossible to obtain significant results.

An alternative for experimentation in environments with this characteristic is the use of Factorial planning technique [14]. Factorial planning makes it possible to measure the effects, or its influences, of one or more variables on the response of a process. This technique consists on identification of a finite set of factors that influences the behavior of the environment, assigning specific and valid values to the levels of each factor, which changes the results of monitored variables. Relationship between factors and levels is exponential, $(levels)^{factors}$, and its most common occurrence is the 2^k Factorial, where a set of k factors assumes two levels of possible values each.

Based on this principle, for the experimental analysis of distributed Mininet solutions, a sequence of three control factors with two variation levels each was identified, forming the representation of a 2^3 Factorial experiment (Fig. 77.3a). First factor corresponds to distributed Mininet technology type ($MininetDistributed$), with levels varying between MaxiNet (MN) and Mininet Cluster with GRE links (MC). Second factor is topology size ($TopologySize$), which corresponds to the amount of pods, in FatTree topology, and cells, in DCell topology, with levels varying between 4 and 12 levels. This variation aims to guarantee the experimental analysis in topologies composed by few and many elements. Third factor is the amount of cluster nodes, varying between 2 and 4 levels, so every environment is tested on a cluster consisting of 2 and 4 physical nodes. Each factor receives a denomination of $x1, x2, \ldots, xn$, that simplifies its identification.

In a planning of a 2^k Factorial experiment, the two levels of each factor are called low level and high level, and can be identified with values (-1), at lower level, and $(+1)$ at higher level, as presented at level column of Fig. 77.3a. With this definition it was possible to identify test combinations of experiment (Fig. 77.3b), whose factors levels distribution definition in the plane followed the procedure::

	factor	level -1	level +1	results		
x1	Distributed Mininet	MN	MC	RAM	proc.	con.
x2	Topology length	4	12	RAM	proc.	con.
x3	Cluster length	2	4	RAM	proc.	con.

(a)

exp.	control factor x1	control factor x2	control factor x3	seq.
1	-1	-1	-1	5
2	+1	-1	-1	7
3	-1	+1	-1	1
4	+1	+1	-1	8
5	-1	-1	+1	2
6	+1	-1	+1	3
7	-1	+1	+1	4
8	+1	+1	+1	6

(b)

Fig. 77.3 Table of factors and planning matrix in 2^3 Factorial design for distributed Mininet solution evaluation. (**a**) Factor table. (**b**) Planning matrix

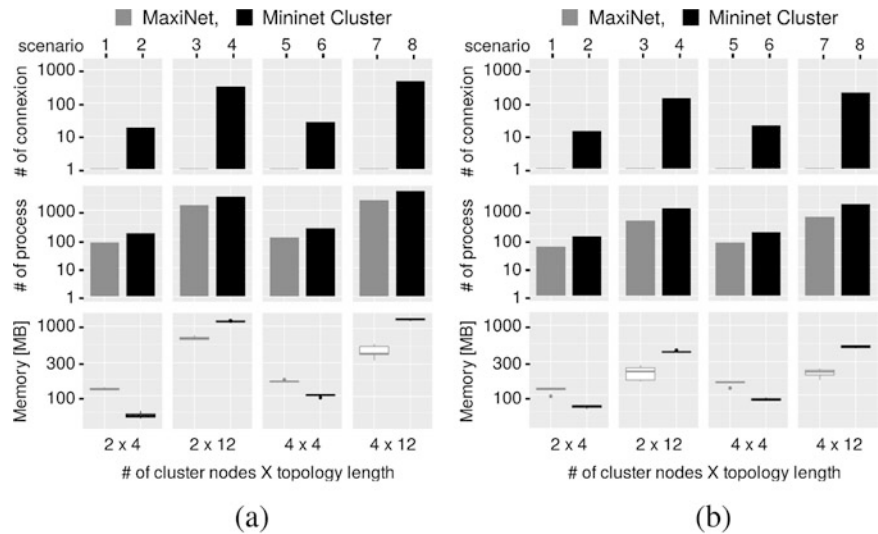

Fig. 77.4 Memory utilization, the number of spawned processes and the number of remote network connections established between cluster nodes in distributed Mininet solution evaluation. (**a**) FatTree topology. (**b**) DCell topology

- For $x1$, column signal of (1) alternates in groups of $2^0 = 1$, that is, continuously.
- For $x2$, column signal of (1) alternates in groups of $2^1 = 2$, that is, in pairs.
- For $x3$, column signal of (1) alternates in groups of $2^2 = 4$, that is, 4 times (-1) followed by 4 times $(+1)$.

Finally, we randomized sequence to execute each experiment test, minimizing possibility of interference of sources not defined in environment. After completing these steps, we executed the experiment.

77.3.4 Environment and Technologies

Cluster used in the experiment consists of 4 servers. Each has 1 processor with 8 cores of 2.40 GHz, 8 GB of RAM and two network interfaces of 1 Gbps each. The S.O. is Ubuntu Server 16.04.5 LTS, kernel 4.4.0-138, and the main programs installed are Mininet 2.3.0d4, MaxiNet 1.2, Open vSwitch 2.5.5 and Pyro4.

Each server was connected in a totally isolated experiment network, consisting of a switch with 1 Gbps interfaces in star topology. This network was used exclusively for exchanging management messages from distributed Mininet solutions in the life-cycle experiment. Second server network interface was connected to a second switch with 1 Gbps interfaces, also in star topology, in a separate network for administration. Physical topology used in cluster nodes interconnection sought to prevent not predicted external factors interference in experiment results.

77.4 Results and Discussion

After definitions presented in Sect. 77.4, evaluation of distributed Mininet solutions were carried out following the sequence defined in Planning matrix (Fig. 77.3b) with a total of 10 replications for each scenario.

77.4.1 Memory, Processes and Connections

Evaluation results presented in Fig. 77.4, show aggregated RAM utilization, number of spawned Linux processes and number of remote network connections created between cluster nodes during FatTree (Fig. 77.4a) and DCell (Fig. 77.4b) Mininet topologies setup.

Results showed that the total amount of used cluster memory varied considerably between all scenarios containing 4 and 12 cells or pods, starting from approximately 55 MB of memory in scenario 2, composed by a FatTree topology and 4 pods distributed over 2 cluster nodes using Mininet Cluster, for up to 1.4 GB of memory used in scenario 8, with the same topology, and 12 pods distributed over 4 cluster nodes using Mininet Cluster (Fig. 77.4a).

There are lower variation in results between scenarios containing the same number of cluster nodes, and pods or cells, varying only Mininet solution. An example of this variation can be observed between scenarios 7 and 8 of DCell topology (Fig. 77.4b). The MaxiNet based scenario used only 218 MB of RAM, while Mininet Cluster used 500 MB of RAM, so Maxinet uses only 42% of memory when compared to Mininet Cluster. This is possible due to the way both technologies manage remote Mininet elements. Maxinet uses Pyro4 to manage remote Python objects (Sect. 77.2.2), so new Mininet elements are requested to Pyro4 server which sends $mnexec$ code to be executed on remote cluster node. Mininet Cluster creates a new SSH connection between the *master*, cluster node where Mininet was called, and *slave*, the node where Mininet element should be created, and this SSH tunnel remains established throughout Mininet element life-cycle. This procedure consumes memory, create processes and make connections between cluster nodes involved in it.

MaxiNet needs Pyro4 daemon running on all cluster nodes even before any local or remote Mininet elements are created. It justifies MaxiNet superior memory usage in scenarios containing few Mininet elements, such as between scenarios 1 and 2 of FatTree and DCell topologies. Pyro4 daemon consumes approximately 55 MB of memory in *FrontEnd* cluster node, and 25 MB of memory in each *Worker* node, while Mininet Cluster does not require daemon service.

Regarding the number of spawned Linux processes and remote connections established between cluster nodes in each scenario, its variation is related to the number of Mininet elements existent in topology, its distribution on cluster nodes, and Mininet solution used. Each Mininet element creates only one process on node where it was created, but as previously showed, Mininet Cluster uses SSH connections to manage remote elements, increasing four new Linux processes and one more network connection between *master* and *slave* cluster nodes for each Mininet element created. MaxiNet uses only few more processes and connections for Pyro4 objects message exchanges. This behavior explains difference of amount processes and connections in scenarios with same number of elements and cluster nodes, varying only Mininet solution, as observed in scenarios 7 and 8 of FatTree and DCell topologies (Fig. 77.4).

77.4.2 Factor Effects

Other important analysis is the effect caused by variation of an individual factor, or a set of them. This analysis helps identify the best strategy for scaling the emulated network topology consuming least cluster resources. Figure 77.5 shows the effect of each independent factor, also known as main effect, interaction effect between the factors evaluated, and a half-normal of effects.

Main effect is analyzed observing factor memory consumption variation, which corresponds to the line slope degree that represents factor when passed from level (-1) to level $(+1)$. As result, it is possible to note the factor with greatest effect in Fig. 77.5a, b is $TopologyLength$, followed by $DistributedMininet$. $Clusterlength$ has little effect on environment memory consumption.

Angle variation between factor lines in Fig. 77.5 identify the interaction effect between a set of factors. Results showed that interaction between $Topologylength$ and $DistributedMininet$ causes oscillation in memory consumption, but this behavior is not observed in $Clusterlength$ interaction with other factors.

To measure the factor effect, or a set of them, on results, we used half-normal effects plot. As an example, it is possible to observe on Fig. 77.5a that value variation in $DistributedMininet$ factor caused a variation of 380 MB of memory consumption, while its interaction with the $Topologylength$ factor caused 400 MB of effect, and $Topologylength$ factor caused 880 MB of memory consumption effect in FatTree topology. This behavior also occurs in DCell topology (Fig. 77.5b).

77.4.3 Setup Time

Another important information to analyze is time taken to execute each action necessary to complete experimentation setup. Figure 77.6 presents aggregation time to complete all experimentation setup actions on scenarios based on 4 cluster nodes, and 4 or 12 pods or cells in topology length. Results showed Mininet Cluster spent more time to complete setup actions than MaxiNet for all scenarios. This behavior is a consequence of time spent by Mininet Cluster to open SSH connections between cluster nodes and send Mininet commands responsible to manage elements and links, as discussed previously. For an example, in FatTree topology scenario with four pods topology length (Fig. 77.6a), Mininet Cluster spent eight times longer than MaxiNet to complete all setup actions.

With the analysis of presented results, it is possible to conclude that Mininet Cluster has inferior behavior to MaxiNet, although the Mininet Cluster is the official imple-

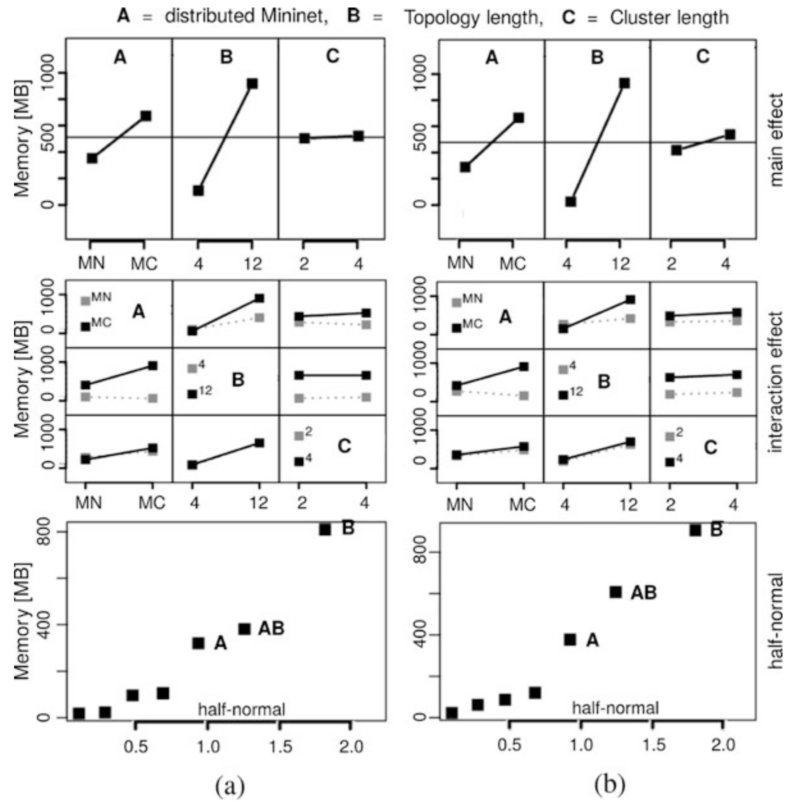

Fig. 77.5 Main effect, interaction effect and half-normal resulting from distributed Mininet solution evaluation. (**a**) FatTree topology. (**b**) DCell topology

Fig. 77.6 Time taken per setup action in distributed Mininet solution evaluation. (**a**) FatTree topology. (**b**) DCell topology

mentation of distributed Mininet solution, MaxiNet manages remote Mininet objects life-cycle in a better way, using Pyro4 name server for it, while Mininet Cluster consumes a lot of processing time and cluster computing resources with SSH connections for container life-cycle management.

77.5 Conclusions and Future Work

In this paper, we presented a comprehensive factorial analysis comparing lightweight and distributed network emulation solutions designed for experimentation. Our approach was to identify key characteristics that make the container-based virtualization lightweight and scalable, suitable for large scale network experimentation. Thus, we perform a number of exploratory experiments on the main distributed emulation candidates solutions: Mininet Cluster and MaxiNet. In order to compare resource consumption on same environment, we implement a 2^k Factorial experimental design to assess the variables influence.

Our results shows a significant advantage of MaxiNet over Mininet Cluster implementation, either in terms of memory consumption, number of spawned processes, number of remote connections established between cluster nodes and time

taken per setup action. Despite of that, Mininet Cluster can be advantageous in non-SDN experiment because it allows remote connection between any kind of element. In terms of future work, we intend to augment the observation variables using other varying hardware resources and also devise an algorithm that could auto-adjust and tune the distributed network emulation from a single machine setup to a cluster, easing the burden for researchers to scale their experiments.

Acknowledgments This work was supported by CAPES - Financing Code 001. Project partially financed by FAPESP (Process no. 2015/24352-9) and Federal Institute of Tocantins (IFTO). Hermes Senger thanks FAPESP for support (Processes 2018/00452-2 and 2015/24461-2) and CNPQ (Process 305032/2015-1).

References

1. Yan, L., McKeown, N.: Learning networking by reproducing research results. SIGCOMM Comput. Commun. Rev. **47**, 19–26 (2017)
2. White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., Joglekar, A.: An integrated experimental environment for distributed systems and networks. SIGOPS Oper. Syst. Rev. **36**, 255–270 (2002)
3. Beshay, J.D., Francini, A., Prakash, R.: On the fidelity of single-machine network emulation in Linux. In: Proceedings of the 2015 IEEE 23rd International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '15), pp. 19–22. IEEE Computer Society, Washington (2015)
4. Huang, T.-Y., Jeyakumar, V., Lantz, B., Feamster, N., Winstein, K., Sivaraman, A. Teaching computer networking with mininet. In: ACM SIGCOMM (2014)
5. Liu, J., Marcondes, C., Ahmed, M., Rong, R.: Toward scalable emulation of future internet applications with simulation symbiosis. In: Proceedings of the 19th International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2015), pp. 68–77. IEEE, Piscataway (2015)
6. Yan, J., Jin, D.: Vt-mininet: virtual-time-enabled mininet for scalable and accurate software-define network emulation. In: Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR '15), pp. 27:1–27:7. ACM, New York (2015)
7. Ganesh, P.I., Hepkin, D.A., Jain, V., Mishra, R., Rogers, M.D.: Workload migration using on demand remote paging. US Patent 8,200,771 (2012)
8. Daniels, J.: Server virtualization architecture and implementation. In: XRDS, vol. 16, pp. 8–12 (2009)
9. Lantz, B., O'Connor, B.: A mininet-based virtual testbed for distributed SDN development. In: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15), pp. 365–366. ACM, New York (2015)
10. Burkard, C.: Cluster Edition Prototype (2014). https://github.com/mininet/mininet/wiki/Cluster-Edition-Prototype. Acessado em 23 June 2018
11. Wette, P., Dräxler, M., Schwabe, A., Wallaschek, F., Zahraee, M.H., Karl, H.: Maxinet: distributed emulation of software-defined networks. In: 2014 IFIP Networking Conference, pp. 1–9 (2014)
12. Al-Fares, M., Loukissas, A., Vahdat, A.: A scalable, commodity data center network architecture. In: Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication (SIGCOMM '08), pp. 63–74. ACM, New York (2008)
13. Guo, C., Wu, H., Tan, K., Shi, L., Zhang, Y., Lu, S., Dcell: a scalable and fault-tolerant network structure for data centers. In: Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication (SIGCOMM '08), pp. 75–86. ACM, New York (2008)
14. Montgomery, D.C.: Design and Analysis of Experiments. Wiley, New York (2006)