Assessing the Limits of Mininet-based Environments for Network Experimentation

David Muelas, Javier Ramos, Jorge E. López de Vergara

Abstract—Virtualization and emulation have become worthy approaches to save humongous amounts of money related to physical resource acquisition expenditures. In this light, the development of network emulation platforms has led a revolution in the research and testing of novel services and applications, as they provide a cost-effective, flexible and reproducible environment for experimentation. However, they present some practical issues: particularly, their scalability is one of the limiting factors as it links the emulated networks that can be successfully deployed on a given hardware. We address this matter by testing the consumption and exploitation of physical resources of one popular network emulation platform, Mininet. We follow a methodology based on the isolation of the threads associated to the Operating System, the virtual hosts, and the monitoring tasks. In such a manner, this approach can measure the effect of the placement of threads in the available cores, and help optimizing bottlenecks that jeopardize the results of network emulations. Additionally, we monitor several Key Performance Indicators for general-purpose Mininet deployments in different network topologies varying the number of active elements, links and network conditions such as packet loss or delay. Our results show that Mininet presents performance bounds in commodity servers that suffice a wide range of general network tests. It achieves aggregated bandwidths above 10 Gb/s and median Round-Trip Time values around 1 ms, even in demanding scenarios where more than a thousand hosts, up to 64-hop paths and 64 subnets are included in the emulated topologies.

INTRODUCTION

NOWADAYS, network operators and developers are highly concerned about how to evaluate and test their future deployments, services and architectures. Traditionally, such tasks have been accomplished using network simulators such as OMNeT++ or ns-3. Nonetheless, the deployment of heterogeneous network elements, the emerging technologies and the users' behavior are usually beyond the capabilities of simulations. Therefore, they fail to provide results close to the real behavior of current networks, which are more non-deterministic and complex [1].

All authors are with the Department of Electronics and Communication Technologies, Escuela Politécnica Superior, Universidad Autónoma de Madrid, Francisco Tomás y Valiente, 11, 28049 Madrid, Spain
E-mail addresses: {dav.muelas, javier.ramos, jorge.lopez_vergara}@uam.es
This work was partially supported by the Spanish Ministry of Economy and Competitiveness and the European Regional Development Fund under the project TRÁFICA (MINECO/FEDER TEC2015-69417-C2-1-R).
Manuscript received September 6th 2017, revised May 8th 2018, accepted August 8th, 2018.
Cite as: D. Muelas, J. Ramos, J. E. López de Vergara, "Assessing the Limits of Mininet-based Environments for Network Experimentation," *IEEE Network*, 2018. DOI: 10.1109/MNET.2018.1700277

This emerging scenario has led network researchers to the exploration of emulation platforms to surpass the limits of network simulations. Using such platforms can improve the experimentation and research in networking in terms of reliability, flexibility, reproducibility and cost. In this line, several tools have been developed to test and design future network architectures and services, even to virtualize and substitute physical infrastructures.

In this context, Mininet [2] has become one of the reference emulation tools, as it integrates several Software Defined Networking (SDN) elements. Mininet, developed at Stanford University, is an open-source, easy-to-deploy and lightweight network emulator that provides a programmable interface to define and build network configurations with virtualized elements. It is intended to alleviate the cost of experimentation and to perform extensive network testing by means of dedicated software. Hence, this system opens the gate to reproducible network experiments [1] and to risk-free experimental environments to test the functionality and performance of novel services [3].

Previous analysis of Mininet capabilities have mainly focused on how it supports experimentation in specific domains —*e.g.*, the study of SDN deployments in [4], [5]. Moreover, behavioral differences among simulation, emulation and physical deployments have been widely explored in the literature —see for instance the results reported by the main Mininet developers[1], or those in [1], [6]. Those works showed that the quality of the measurements in these *softwarized networks* allows replacing the previous evaluation and testing platforms —if slight divergences with respect to physical deployments are admissible [7].

However, resource consumption and performance degradations due to scheduling of execution threads have not been extensively analyzed, despite the direct effect that they can exert on experimental results —in fact, and to the best of our knowledge, only some recent works have considered the scalability of such platforms with focus on SDN controllers, as shown in the comprehensive survey in [8]. Thus, this work tries to shed light upon these matters by evaluating the physical requirements and behavior of Mininet during the emulation of network topologies in an off-the-shelf physical server using different isolation strategies for physical resources. Our evaluation considers a set of scenarios that aims to provide an exhaustive character-

1. https://github.com/mininet/mininet/wiki/Publications

ization of Mininet performance in terms of the number of deployed elements (namely, switches and hosts) and link characteristics (specifically, packet loss, delay and jitter), to represent general emulation setups. Our evaluation also takes into account general network Key Performance Indicators (KPI), to assess the viability of up to 1500-hosts network deployment emulations in a single physical server.

Such an approach lets us extract several insights into the behavior of Mininet-based network emulations. On the one hand, we evaluate how different thread placement strategies (i.e., CPU isolation and affinity) and network conditions (namely, packet loss and delay) shape the resource consumption, and which bottlenecks may appear because of suboptimal policies. This analysis also reveals which are the statistical main effects and interactions of the number of hosts and switches in the resource consumption —hence, showing the effect of varying them independently and simultaneously. On the other hand, we introduce different generic network loads (namely, massive data transmission, data encryption with VPN tunneling and switching/routing) in controlled environments, to characterize how they affect the overall achievable performance. We believe that these results are useful for the early stages of the design and deployment of tests in emulation environments, hence potentially improving the research, testing and innovation in the area.

To present our findings, we first analyze the architecture and operation of Mininet to extract key guiding principles for the setup of experiments. Then, we detail the experimental design, with a complete description of both the set of scenarios under test and hardware and software considerations. After that, we thoroughly analyze the results, and discuss their implications. Finally, we highlight the main conclusions and point to future directions of this work.

INSIGHTS INTO MININET DEPLOYMENTS

Architecture and operation

In general, network emulation platforms encompass three types of functional blocks: hosts, switching elements, and links. Memory and CPU usage are of paramount importance in emulation environments, given that virtual elements make extensive use of such physical resources. Consequently, most emulation platforms apply lightweight virtualization approaches to represent network nodes and improve scalability.

Specifically, Mininet uses Linux network namespaces to emulate multiple network stacks inside a single physical system, whereas other state-of-the-art proposals (such as VNX[2], CORE[3] or IMUNES[4]) make use of other heavier containers —e.g., Docker or LXC (LinuX Containers). Regarding switching elements, Mininet relies on the capabilities of external tools such as Open vSwitch (OvS) or Indigo Virtual Switch.

By default, Mininet uses OvS to transmit traffic between the emulated hosts. OvS tasks are distributed among several processes, which run both in kernel and user spaces. Specifically, at kernel level several *datapaths* (similar to bridges) can be defined. Each *datapath* comprises a set of virtual ports (vPorts) and a flow table. Such flow table is used to forward incoming packets or execute specific actions (e.g., dropping a packet) based on their flow-tuples. When a packet does not match any existent flow, it is forwarded to a process executing in user space, which typically acts as an OpenFlow controller and inserts a new flow defining the behavior of the kernel *datapath* for such packet.

Since its first versions, Mininet provides a Python API[5] which manages GNU/Linux `cgroups` to deploy and operate all these network elements. This approach lets users limit and assign certain physical resources to each network element. However, if the existing physical resources are insufficient, the achievable performance may be far from the theoretical maximum [9].

Typical use cases, which comprise new protocols and application testing and SDN controller development [10]–[12] or Virtual Data Centers and Virtual Networks (VNs) [13], impose requirements not only driven by CPU or memory but also by available bandwidth, packet loss or delay. That is, processes under test may require a baseline scenario where both a minimum amount of bandwidth and a maximum delay are guaranteed to a large number of hosts, rendering useless otherwise. Hence, the overhead of virtualization and hardware bottlenecks during the scheduling of execution threads must also be considered during the evaluation of these platforms, to assure their suitability.

This operationalization exposes two capital matters for the analysis of network emulation platforms. First, the expected resource consumption of two different groups of tasks —namely, Operating System (OS) and OvS tasks in both kernel and user space, and Mininet threads, respectively. Second, the degradation of maximum achievable system performance because of the virtualization layers and number of active elements.

Resource consumption and performance

With the aforementioned matters as starting point, we state some expectable aspects about the resource consumption and performance of Mininet deployments prior to the experimental design of this study.

First, we have to consider how the intrinsic properties of the network affect the consumption of resources. In this sense, the evaluation must assert whether it depends jointly on the number of hosts and switches in the network, and how it grows when they are increased. Moreover, the relation between this consumption and link characteristics (i.e., packet loss, delay and jitter) is also a relevant issue, as they result useful in many scopes.

Second, we focus on how can resource scheduling be optimally adapted. In this light, we analyze if the expected CPU loads exhibit different patterns between cores executing OS and OvS tasks, and cores executing Mininet specific tasks taking into account their behavior when emulated network characteristics change.

Finally, the degradation of the maximum achievable system performance may prevent the successful exploitation of Mininet in some scopes. Hence, we assess if, despite of the sharing of resources, Mininet can provide expected

2. http://www.dit.upm.es/vnx
3. http://www.nrl.navy.mil/itd/ncs/products/core
4. http://imunes.net

5. http://mininet.org/api/annotated.html

network performance baselines above the requirements for the emulation of typical network deployments.

Regarding physical resource consumption, the first aspect aims to expose which factors can produce hardware limitations when deploying network topologies in Mininet with an increasing number of active elements [14] and different link characteristics. On its part, the second one is intended to detect how CPU usage differs among cores executing tasks in the previously distinguished groups. Moreover, given the heterogeneity of the possible applications to be deployed in an emulated scenario, we propose the evaluation of the physical resource utilization in a best-case situation, with low activity in the virtual hosts. The last aspect focuses on the achievable network performance when using Mininet. We have selected as illustrative network KPIs the Bulk Transfer Capacity (BTC) and the Round Trip Time (RTT), to evaluate possible burdens in emulations with thousands of hosts. Interestingly, these network KPIs are upper bounded by the maximum system performance, so this approach exposes its degradation due to virtualization and addition of active network elements.

EXPERIMENTAL DESIGN

Keeping in mind the aspects above, we have envisaged an experimental design, which aims to obtain a comprehensive characterization while preventing bias from specificities. To do so, our design is rooted on *(i)* low network load during the resource consumption tests; *(ii)* complete availability of resources for network emulation during the performance tests; and *(iii)* basic topological structures for virtual networks. Although this may seem somehow optimistic, these conditions allow the extraction of general figures and trends with minimal affection of specific applications and spurious interactions with uncontrollable factors. To present its details, we first describe the scenarios under test, and then we elaborate on the hardware and software setup of the system.

Scenarios

We defined the set of scenarios in Figure 1 to measure resource consumption and analyze the operating point of Mininet in terms of intrinsic characteristics of the Virtual Networks (VN):

- VN_1 consists of N hosts connected through M switches. Hence, it is a multi-level tree topology where leaves contain hosts connected by different switches in a layered mode —i.e., only switches 1 and M are connected to end hosts. When $M = 1$, this scenario corresponds to a single star topology, typical in simple data center architectures or LANs.
- VN_2 represents a common architecture to provide dedicated IPSec tunnels to N hosts. VN_2 is quite similar to VN_1 with $M = 1$, but it includes an IPSec server between the hosts and the external network access (C).
- VN_3 is composed of M different subnets connected through a core switch. Each subnet contains N hosts. In this scenario, OpenFlow rules (based on packet destination subnet) are installed in switches S_1 to S_M and in the core switch to provide a routing alike behavior. This scenario reflects emulations of large enterprises and small ISPs, and data centers where hosts in the same rack are connected by a Top-of-Rack (ToR) switch and the next levels encompass switches that interconnect different racks.

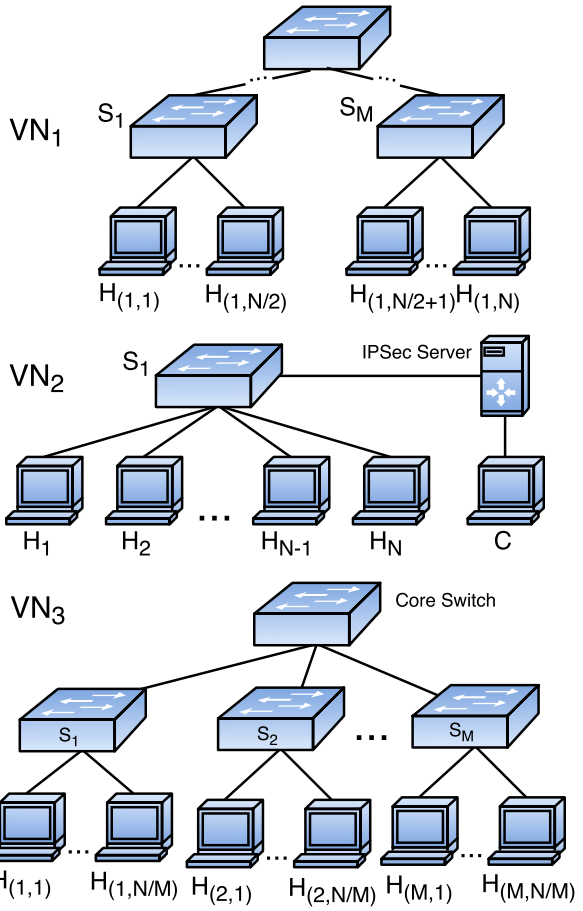


Figure 1. Diagram of the virtual networks considered in our experiments.

VN_1 aims to evaluate the effects of the number of active elements for VNs deployed in a physical server. VN_2 is used to characterize performance degradation when data is not only switched but also encrypted —thus, exploring performance when more sophisticated tasks are executed in the emulated scenario. VN_3 is designed to evaluate the impact of OpenFlow rules on the network performance when the number of subnets varies —the total number of hosts is kept constant, and they are distributed uniformly across subnets in each trial to prevent spurious effects.

We used OpenFlow rules to discard possible effects related to the implementation of third-party elements. Such rules are installed with the `ovs-ofctl` command at topology start-up time, so they are not modified during the experiment. For the sake of simplicity and to avoid emulating ARP proxies, ARP tables are statically configured on each host.

Hardware and software setup

We have executed all the experiments in a commodity server with two Intel Xeon E5-2620 v2 processors (6 cores per processor) running at 2.10 GHz without Hyper-Threading, and 32 GB of RAM. The server ran Ubuntu 14.04.1 with a Linux kernel 4.4.0-45. We used Mininet 2.2.2[6] with its default configuration. We deployed the previously described scenarios using the Python API that this version provides, and we configured the network elements to customize their pinning to physical cores with the options in the API.

We used `isolcpus`[7] kernel parameter to isolate OS/OvS threads from Mininet and resource consumption monitoring threads, minimizing their mutual interaction when characterizing the system load. Additionally, we used the `taskset`[8] Linux command to pin threads manually to specific cores. We assessed that there is no significant load when experiments are off (namely, less than a 5%), thus providing a good estimation of the CPU load related to OvS and Mininet activities.

To characterize the physical resource consumption, we monitored the system during several deployments of VN_1 with variable numbers of both hosts and switches and different link configurations —packet loss, delay and jitter. The number of both hosts and switches spanned from 4 to 1024. Packet loss was varied to obtain an end-to-end rate ranging from 0 to 10%, and delay was configured to three different average levels (0, 50 and 75 ms) and two values of jitter —0 and 10 ms. Link configuration relied on the available options in the API of Mininet, which uses `tc` and `netem`[9].

In each trial, `ping` was executed 15 seconds while retrieving CPU and memory usage statistics —2 samples per second, using `/proc/stat` and `free` command to gather CPU and memory measurements, respectively. Each experiment was repeated 50 times, providing averaged results with their standard errors —this provides an idea of the deviation of the sample mean with respect to the population mean, thus quantifying the convergence to the actual expected system behavior.

Regarding the CPU usage, we analyzed two different placement strategies for OS/OvS threads —i.e., pinning execution threads to specific isolated cores. In the first one, we isolated all physical cores except the first one, as some OS threads are statically bound to that core. In such core, both OS and OvS tasks are executed. Regarding the remaining cores, 6 out of 11 are used to distribute Mininet tasks uniformly. CPU and memory monitoring processes are scheduled to be executed in the remaining unused cores to avoid interference. In the second strategy, we isolated all physical cores except the two first, which are used by OS and OvS. The remaining cores are assigned as in the first strategy. These two scheduling configurations illustrate how OvS scales with the number of available cores.

6. git://github.com/mininet/mininet
7. https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/cpu-partitioning/isolcpus
8. https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/cpu-partitioning/taskset
9. https://wiki.linuxfoundation.org/networking/netem

We also assessed the RTT and the BTC to evaluate the network performance. First, we conducted throughput measurements with TCP by running `iperf` in VN_1, to link the degradation of the system maximum performance with respect to topology size. TCP provides enhanced performance results in this setup due to the use of Super Jumbo Frames (SJF) [15] and TCP offloading mechanisms that allow sending IP datagrams of up to 65535 bytes —such behavior is observed by default when running measurement tools such as `iperf`[10]. After these measurements, we evaluated RTT with `ping` when traffic is not only forwarded among hosts, but also transformed. To do so, we established IPSec dedicated tunnels in VN_2 using the Linux utility `strongSwan`[11]. In VN_3 we performed both bandwidth and RTT measurements as in the other virtual networks and varying the number of subnets. For all these tests, we assessed again that the idle system load was below 5% and without core isolation to maximize the utilization of the server. Additionally, we considered the default TCP parameters —window sizes of 85.3 KBytes and 2.5 MBytes for server and client, respectively.

Analysis of Results

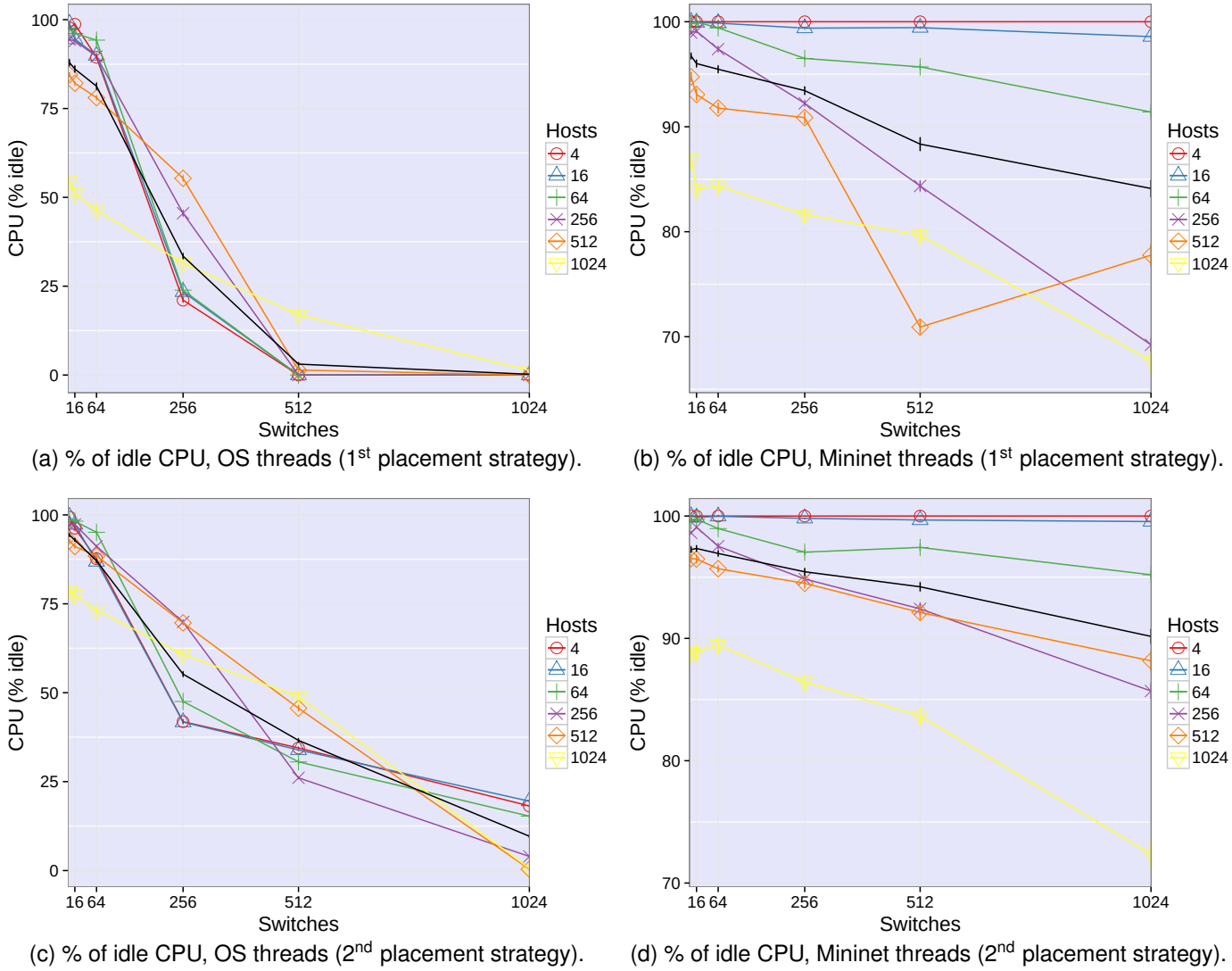
Topology size and resource consumption

Figures 2 and 3 present the measured free CPU and memory, respectively —i.e., the available resources for other applications in the physical host. Each graph represents the complementary of the average observed utilization with standard errors as error bars. The number of switches is included as independent variable, while each colored line with markers corresponds to a different number of hosts. The black line with no markers corresponds to the averaged results without partitioning the results by number of hosts. This representation allows a qualitative analysis of the effect of jointly adding hosts and switches in the resource consumption of Mininet-based deployments. Regarding the presented data, figures 2a and 2b are obtained with the first placement strategy (only one core available for OS/OvS tasks), whereas figures 2c and 2d include the measurements with the second one —two cores available for OS/OvS tasks.

The scalability of OS and OvS threads is assessed by comparing figures 2a and 2c. Remarkably, even in a context without heavy activity in the active elements, a single core cannot cope with traffic forwarding if the number of switches is greater than 512. On the counterpart, the assignment of an additional core is sufficient to support the activity of deployments with 1024 switches and 1024 hosts. In both cases, the main effect of the number of switches turns up as the dominant factor as the number of hosts only produces remarkable effects for less than 512 switches.

On the other hand, figures 2b and 2d show that the number of hosts produces the statistical dominant effect in the resource consumption of Mininet processes —*e.g.*, see the almost constant percentage of idle CPU when varying the number of switches with 4 hosts. Additionally, the interaction of hosts and switches is only relevant for more than 512 switches, as exposed in Figure 2d. However, this

10. https://linux.die.net/man/1/iperf
11. https://www.strongswan.org/



(a) % of idle CPU, OS threads (1st placement strategy).

(b) % of idle CPU, Mininet threads (1st placement strategy).

(c) % of idle CPU, OS threads (2nd placement strategy).

(d) % of idle CPU, Mininet threads (2nd placement strategy).

Figure 2. Percentages (mean and standard error) of idle CPU varying the number active elements in VN_1, and the number of CPUs assigned to each task with two different thread placement strategies. Black line with no markers is the average aggregating all number of hosts.

effect is less noticeable in Figure 2b because the core running OS/OvS threads is fully saturated, producing bottlenecks that distort the behavior in these latter cores.

Unsurprisingly, the memory consumption does not depend on the task scheduling strategies nor the number of used cores. Figure 3 presents the changes in the memory footprint for each scenario, and confirms that both the main and interaction effects of the number of hosts and switches are significant to explain the memory consumption.

Link characteristics and resource consumption

Once we have characterized how physical resource consumption is tied to the topology size, we focus on how link conditions affect the hardware requirements. The results, using the second placement strategy are presented in Figure 4. In this case, we only considered the second placement strategy, as the previous results exposed that the first one does not suffice to deploy all the topologies without saturating the system.

Figure 4 simultaneously displays series of boxplots to characterize the overall behavior regarding each level of packet loss or delay, hence providing an idea of the distribution of values. Additionally, it also includes the mean value for the groups defined in terms of active elements as dots, which color and size indicate the number of hosts and switches respectively —we added small fluctuations to the independent variable, to reduce overlapping and improve visualization.

In figures 4a and 4b we assess how CPU utilization varied depending on the percentage of lost packets for cores executing OS/OvS tasks, and those devoted to Mininet threads. Both groups exhibited similar affection by the introduction of packet loss, with an increased overall idle percentage and higher variability.

On the other side, the measurements with additional delay and jitter included in figures 4c and 4d showed higher levels of CPU utilization, as well as higher variability.

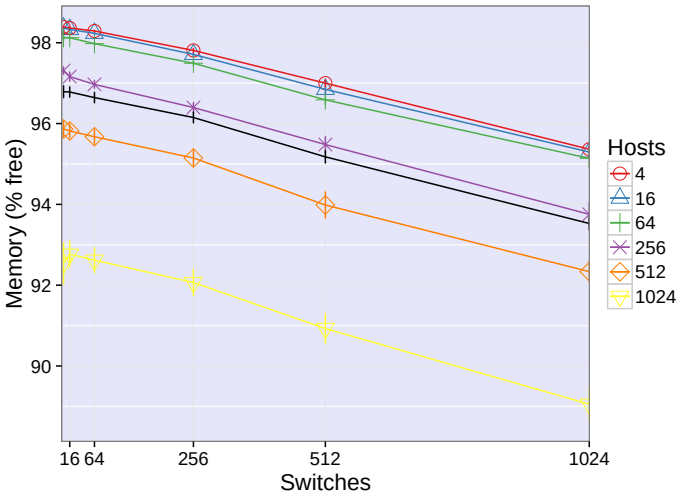


Figure 3. Free memory (mean and standard error) when varying the number of active elements in VN_1. Black line with no markers is the average aggregating all number of hosts.

Finally, memory consumption presented similar trends, with low affection related to packet loss and higher footprints in the case of additional delay.

This behavior is not surprising. On the one hand, packet loss entails less packets to be processed and forwarded both at the switching elements and at receiver hosts. On the other hand, higher delay and jitter leads to the same amount of packets to be processed but *(i)* with additional processing in the links introducing thus delay and *(ii)* demanding more processing time in the sender side of the `ping` probes.

Network performance

First, we conducted experiments varying the number of hosts from 10 to 1500 in VN_1. To obtain the upper boundary for the achievable bandwidth, we first defined a simple scenario with two hosts directly connected through a switch and one `iperf` process running. The result of such experiment provided the upper bound of the achievable aggregated bandwidth, which in our case is located near 50 Gb/s —similar to the average best case reported by the `mbw`[12] tool for memory bandwidth measurement.

To cover all possible client-server distribution cases, for each number of hosts (N), we vary the number of servers from 1 to N while keeping the number of clients constant to N. Figure 5a shows the boxplot representation of bandwidth reported by `iperf` servers —30 samples each, one per second of the experiment. Remarkably, the median achievable bandwidth decreases as the number of hosts increases. The measurements exposed that several hosts stop transmitting during short time periods when the number of hosts is large, due to the congestion of the shared links and to synchronization issues in the scheduling of processes. These facts also explain both the increasing overall variance and dispersion of outliers when the number of hosts grows above a certain threshold —see the trends in the case of 100 hosts. The median is located near 15 Gb/s of aggregated BTC in the worst-case scenario (1500 hosts), which is a fairly good result.

After that, we evaluated how the number of switches affects the achievable bandwidth. In this case, the number of hosts (N) remained constant at 100 while we perform `iperf` measurements varying the number of switches. Figure 5b includes the boxplots of the achievable bandwidth in VN_1 when varying the number of switches. Observing the results, we can state that adding up to 10 switches to our topology does not significantly modify the achievable bandwidth, which median value remains stable near 22-23 Gb/s of aggregated BTC.

Next, we analyzed the variation of RTTs when adding hosts to VN_2. All the hosts were connected to C using an IPSec tunnel through an intermediate IPSec server to provide a more complex scenario. The median RTT is located below 1 ms for 1000 or fewer hosts, and in the case of 1500 hosts, it is near 1 ms. Even in this latter scenario, the samples above 100 ms represent less than a 2% of the measurements, and packet losses were not observed.

To conclude, we evaluated the impact of adding OpenFlow rules on switches, both for the aggregated bandwidth and RTT values. Figure 6 includes the measurements of those KPIs in deployments of VN_3 with 1024 hosts. Regarding aggregated bandwidth, the trends tied to the number of subnets show performance improvements because of the trade-off among the number of subnets and their size. On the other hand, RTT increases with the number of subnets with maximum values around 1 ms.

KEY REMARKS AND DISCUSSION

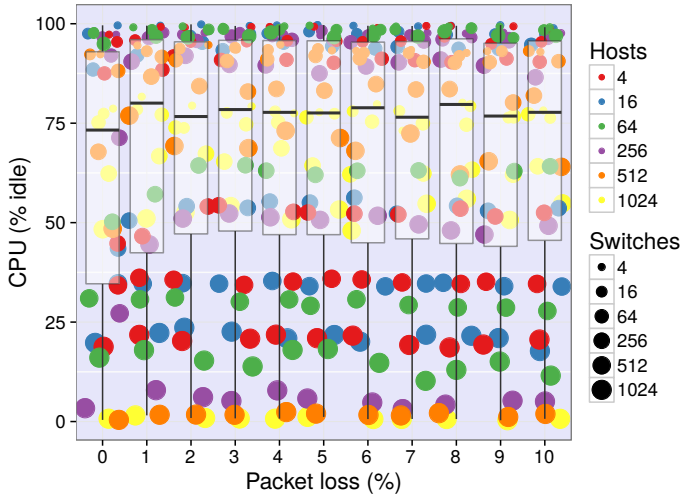
The aforementioned results expose several practical issues that users of Mininet should take into account:

- The main and interaction statistical effects of network elements as explanatory variables for CPU load show that utilization differs between the cores dedicated to the OS/OvS threads, and those isolated for Mininet tasks in the absence of computationally-intensive processes. Additionally, there are significant relations between the CPU load in both groups when the cores devoted to the OS/OvS are saturated.
- The median aggregated bandwidth is above the 50% of memory maximum bandwidth. Those figures provide an estimation of the bandwidth for the flows that are generated from each host.
- High latency values are rarely reported. This enables the use of sophisticated middleboxes beyond simple switching/routing elements, without significant network performance degradation.

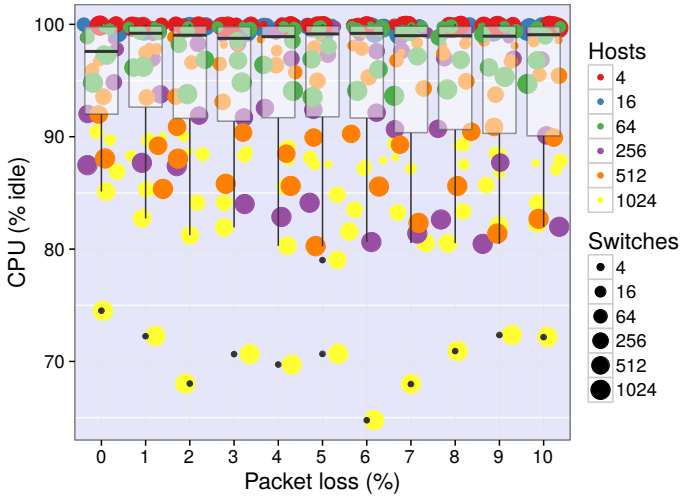
Consequently, we can define a set of rules to improve the results of Mininet-based experimentation:

- CPU affinity and isolation should be carefully tuned to prevent artificial bottlenecks.
- Network performance is tied to memory bandwidth and scheduling of processes. Therefore, baseline measurements must be accomplished to prevent biases by system overloads, taking into account the significant factors and interactions extracted above.

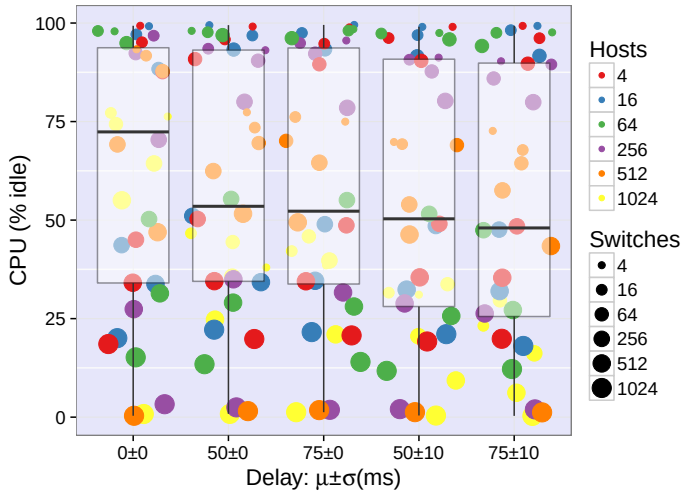
12. http://manpages.ubuntu.com/manpages/trusty/man1/mbw.1.html



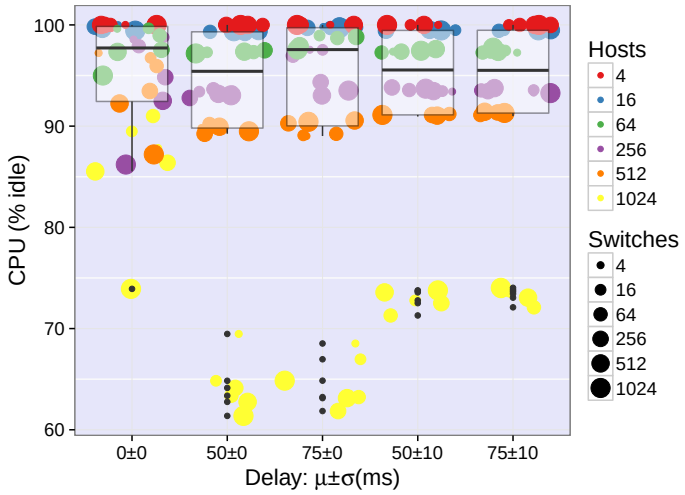
(a) % of idle CPU, OS threads (2nd placement strategy and varying packet loss).



(b) % of idle CPU, Mininet threads (2nd placement strategy and varying packet loss).



(c) % of idle CPU, OS threads (2nd placement strategy and varying delay).



(d) % of idle CPU, Mininet threads (2nd placement strategy and varying delay).

Figure 4. Boxplots of idle CPU varying packet loss (4a and 4b) and delay (4c and 4d) in VN_1. Average values for each network size (i.e., hosts and switches) are presented with dots.

- Delay and jitter affect physical resource consumption, so they must be considered in the hardware requirements. On its part, packet loss exhibits an almost negligible effect.

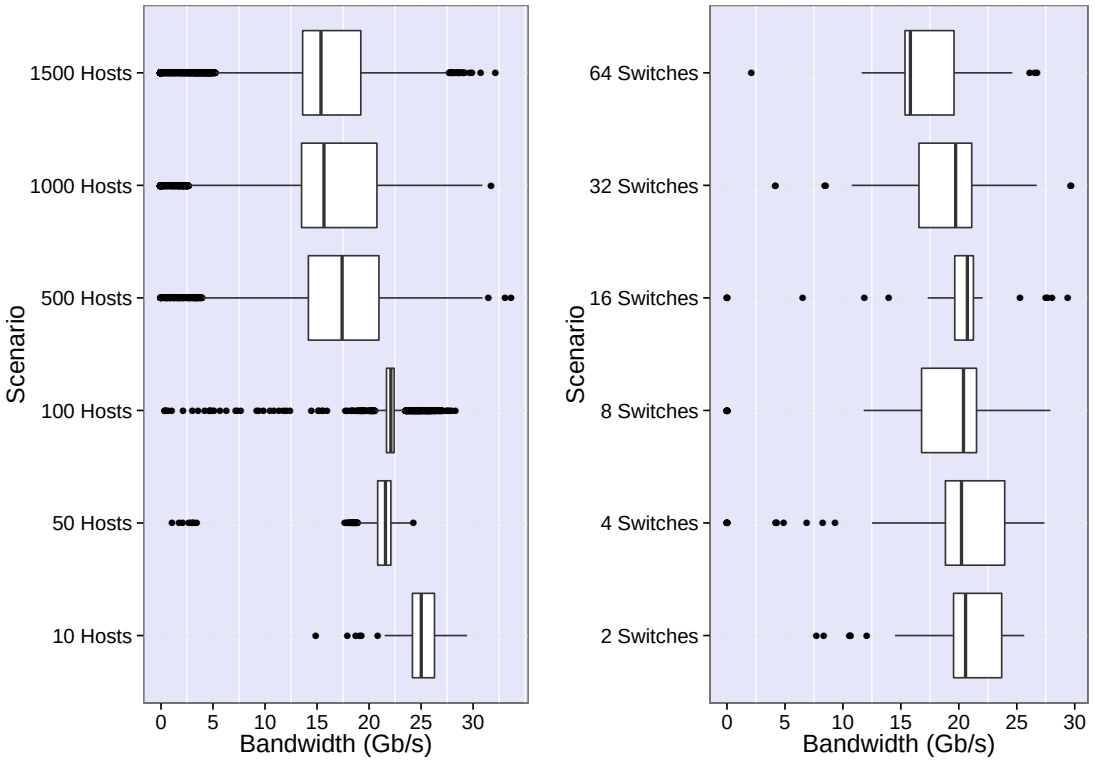
Our characterization of the resource consumption and network performance of Mininet provides a basis for the deployment of more complex and realistic networking scenarios. We believe that our results can help researchers to test network deployments in a more cost-effective manner. Specifically, Mininet-based experimentation solves key problems of physical testbeds (by reducing the expenditures in hardware, energy consumption and operation) and minimizes the risk of failure or anonymity constraints that appear when testing real software and implementations in actual networks.

CONCLUSIONS

We have presented a characterization of the limiting factors and possible bottlenecks for Mininet deployments. We have assessed the performance of this tool for general network emulations with up to 1500 hosts by controlling the different network elements involved in each particular scenario and forcing the pinning of threads to specific isolated cores.

With such a methodology, we have studied the physical requirements (i.e., CPU usage, number of cores, and memory consumption) distinguishing among OS/OvS and Mininet tasks while varying the principal network elements (namely, hosts, switching elements and subnets) and conditions —delay and packet loss.

Our experiments also covered the behavior of aggregated bandwidth and RTT, which presented baseline median values around 15 Gb/s and near 1 ms, respectively. Furthermore, aggregated bandwidth values below 10 Gb/s



(a) Variable number of hosts, with 1 switch. (b) Variable number of switches, with 100 hosts.

Figure 5. Bandwidth characterization in VN_1.

and RTT values above 100 ms are rarely reported, which proves that Mininet is a robust platform to perform a wide range of network experiments and tests.

Finally, the scripts to accomplish the experiments in this paper are open as complementary material to extend the tutorial nature of this work.[13] With this, we provide researchers and practitioners with a set of results and tools to decide if Mininet suits their requirements when carrying out network experiments.

While we believe that these results are worthwhile to report themselves, we envision some future directions of this work. We point to the evaluation of distributed Mininet-based deployments and other network emulation and virtualization platforms such as the aforementioned ones, to compare them quantitatively. Additionally, performance aspects beyond network operations (*e.g.*, hard disk access performance degradation) should be studied to completely characterize Mininet behavior for other use cases.

References

[1] D. Pediaditakis, C. Rotsos, and A. W. Moore, "Faithful reproduction of network experiments," in *Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '14. New York, NY, USA: ACM, 2014, pp. 41–52.

[2] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for Software-Defined Networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, pp. 19:1–19:6.

[3] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. New York, NY, USA: ACM, 2012, pp. 253–264.

[4] S.-Y. Wang, "Comparison of sdn openflow network simulator and emulators: Estinet vs. mininet," in *2014 IEEE Symposium on Computers and Communication (ISCC)*, June 2014, pp. 1–6.

[5] I. Z. Bholebawa, R. K. Jha, and U. D. Dalal, "Performance Analysis of Proposed OpenFlow-Based Network Architecture Using Mininet," *Wireless Personal Communications*, vol. 86, no. 2, pp. 943–958, 2016.

[6] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Mininet Performance Fidelity Benchmarks," 2012.

[7] J. Yan and D. Jin, "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*, ser. SOSR '15. New York, NY, USA: ACM, 2015, pp. 27:1–27:7.

[8] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN)," *Computer Networks*, vol. 112, pp. 279 – 293, 2017.

[9] J. Ramos, P. Santiago del Río, J. Aracil, and J. López de Vergara, "On the effect of concurrent applications in bandwidth measurement speedometers," *Computer Networks*, vol. 55, no. 6, pp. 1435 – 1453, 2011.

[10] M. Gupta, J. Sommers, and P. Barford, "Fast, Accurate Simulation for SDN Prototyping," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 31–36.

[11] K. Kaur, J. Singh, and N. S. Ghumman, "Mininet as Software Defined Networking Testing Platform," in *International Conference on Communication, Computing & Systems (ICCCS)*, 2014.

[12] S. Salsano, P. L. Ventre, L. Prete, G. Siracusano, M. Gerola, and E. Salvadori, "OSHI - Open Source Hybrid IP/SDN Networking (and its Emulation on Mininet and on Distributed SDN Testbeds)," in *Software Defined Networks (EWSDN), 2014 Third European Workshop on*, Sept 2014, pp. 13–18.

13. https://github.com/hpcn-uam/MininetTesting

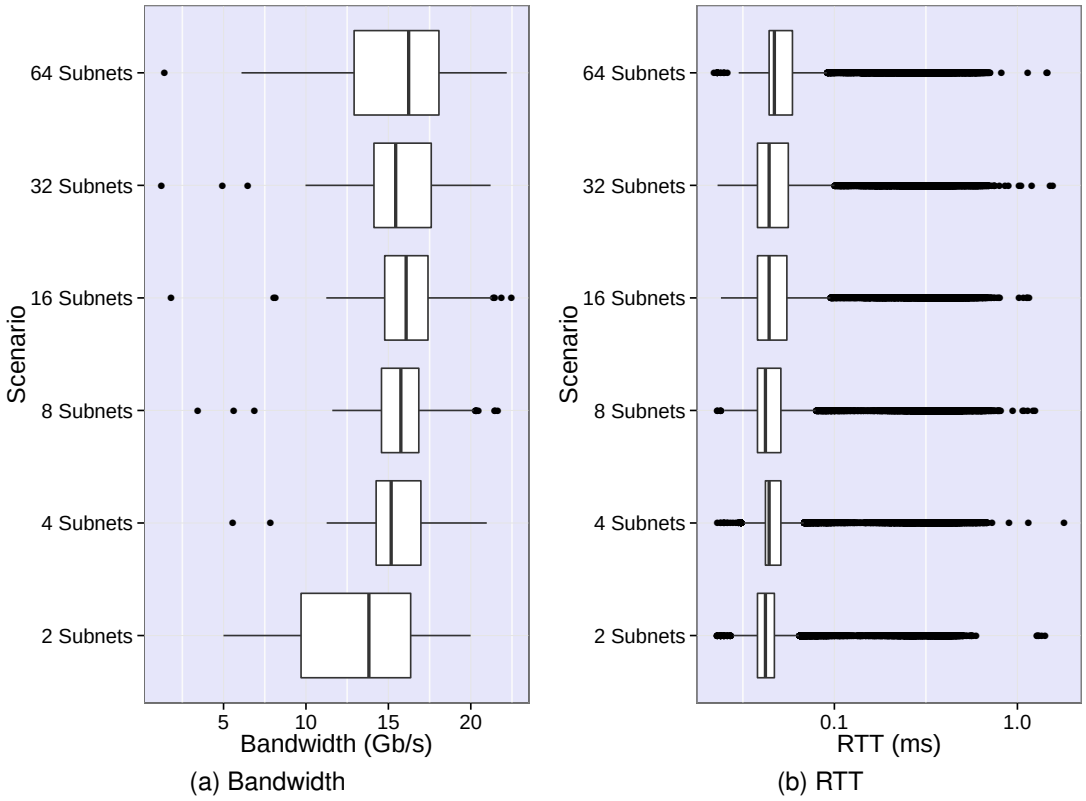


Figure 6. Bandwidth and RTT varying the subnets in VN_3.

[13] S. Liu, H. Xu, L. Liu, W. Bai, K. Chen, and Z. Cai, "Repnet: Cutting latency with flow replication in data center networks," *IEEE Transactions on Services Computing*, pp. 1–1, 2018.

[14] M. Martinello, M. R. N. Ribeiro, R. E. Z. de Oliveira, and R. de Angelis Vitoi, "Keyflow: a prototype for evolving SDN toward core network fabrics," *IEEE Network*, vol. 28, no. 2, pp. 12–19, March 2014.

[15] W. Rutherford, L. Jorgenson, M. Siegert, P. Van Epp, and L. Liu, "16000-64000 B pMTU Experiments with Simulation: The Case for Super Jumbo Frames at Supercomputing '05," *Opt. Switch. Netw.*, vol. 4, no. 2, pp. 121–130, Jun. 2007.

David Muelas (dav.muelas@uam.es) received the degrees in Mathematics and Computer Science in 2013, and M.Sc. degrees in Mathematics and Applications, and in Information and Communications Technologies in 2015, all of them from Universidad Autónoma de Madrid (Spain). Currently, he is a Ph.D. student in that university, and collaborates as research assistant in the HPCN group. His research interests are on the analysis of network traffic, software defined networks and applied mathematics.

Javier Ramos (javier.ramos@uam.es) received the M.Sc. degree in computer science and the Ph.D. degree in computer science and telecommunications from the Universidad Autónoma de Madrid (Spain), in 2008 and 2013, respectively. Currently he works as lecturer at Universidad Autónoma de Madrid. His research interests are on the analysis of network traffic, quality of service, software defined networks and network function virtualization.

Jorge E. López de Vergara (jorge.lopez_vergara@uam.es) is associate professor at Universidad Autónoma de Madrid (Spain), and founding partner of Naudit HPCN, a spin-off company devoted to high performance traffic monitoring and analysis. He received his M.Sc. and Ph.D. degrees in Telecommunication Engineering from Universidad Politécnica de Madrid (Spain) in 1998 and 2003, respectively. He researches on network and service management and monitoring, having co-authored more than 100 scientific papers about this topic.