Reproducing Network Experiments in a Time-controlled Emulation Environment

Diana Andreea Popescu
Computer Laboratory
University of Cambridge
Email: diana.popescu@cl.cam.ac.uk

Andrew W. Moore
Computer Laboratory
University of Cambridge
Email: andrew.moore@cl.cam.ac.uk

***Abstract*—Network emulation environments are a valuable research tool that offer the possibility of conducting complex network experiments and can be seen as an enabler of experimental reproducibility. Keen to quantify this approach, we reproduce experiments presented in QJump [1] in the time-controlled network emulation framework, called SELENA [2]. We first describe in detail how we built the experimental setup. Next, we evaluate the SELENA emulation framework using real world applications, Hadoop MapReduce, memcached and PTPd. We demonstrate our findings, presenting the differences between the results obtained on a hardware testbed and in the emulation environment.**

***Index Terms*—Network Experiments, Reproducibility, Time Dilation, Network Emulation.**

I. Introduction

When trying to reproduce the results presented in a network systems research paper, we are faced with many challenges: critical details missing, original source code unavailable or unclear experimental setups. The practice of making code and experimental procedures available is merely the beginning. Reproducible network experiments should be the norm within the networking community. Despite recent efforts in open sourcing the code and experimental results [3], interest in the area of experimental reproducibility remains low. We evaluate a promising approach permitting *canning* of entire experiments as virtual machines. Wishing to assess the value of the time-controlled network emulation framework, SELENA [2], we set out to reproduce the network experiments used in the QJump project [1], based on the open source code and data provided by the authors [3]. We reproduce their results both on a hardware testbed and in SELENA, and we explore the limits of experimental reproducibility for our chosen complex network experiments. Our motivation for this work is twofold: i) evaluating SELENA as a tool for running complex experiments, and ii) providing an independent validation of the QJump project, as a case study. We make the following contributions:

- We present a reproduction of complex network experiments permitting wide reuse, using the network emulation framework SELENA on top of which we set up the experiments used to evaluate the QJump [1] project.
- We quantify the practical tradeoff between experimentation time and accuracy of reproduction in time dilation enabled emulation frameworks.
- We perform experiments to understand the limits of network emulation in terms of reproducing network experiments, focusing our measurements on latency and clock offset.

The rest of the paper is organized as follows. Section 2 offers details about the SELENA framework. Section 3 explains the QJump project. Sections 4 and 5 describe the experimental setup and the experiments that we ran, whereas section 6 discusses our experimental results. We give an overview of related work in section 7 and we conclude in section 8.

II. The SELENA framework

A. Architecture

We provide a brief overview of the network emulation framework SELENA [2]. SELENA employs the technique of time-dilation to slow down the progression of time in order to overcome the scalability limitations of network emulation. It uses OS-level virtualization and is built on the open-source hypervisor Xen [4]. Each host or network device is mapped to a VM, while each link is mapped to a pair of virtual network interfaces bridged in Dom0 (the host domain, which is the initial domain started by the Xen hypervisor on boot). Figure 1 presents a simple network topology with 2 hosts and one switch, which are mapped to different VMs, and the network links between the switch and the two hosts, consisting of the bridges between the guests' virtual network interfaces. A virtual network interface in Xen consists of a pair of devices: the frontend device in green color and the backend device in pink color as seen in Figure 1. The frontend device resembles a physical Ethernet NIC in the guest domain and under Linux it is bound to the *xen-netfront* driver [5]. The backend device (the *xen-netback* driver) is transparent to the guest and is interconnected with the frontend device using shared memory rings [2].

SELENA provides a Python API that allows one to define an experimental scenario consisting of the network topology and the network parameters (link capacity and latency characteristics). The scenario is deployed by SELENA, which uses the *xen-api* API to create the VMs defined in the experiment and the bridges between the VMs in Dom0. Each VM needs to run an agent that communicates with a central communication

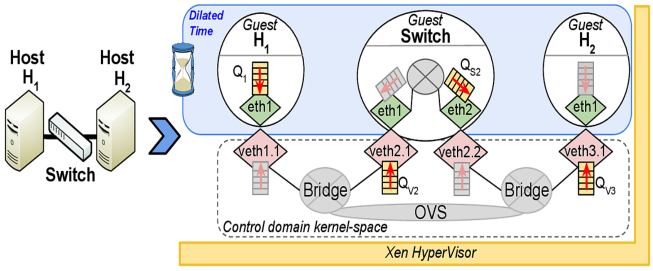

Fig. 1. SELENA emulation architecture built on Xen [2].

service running in Dom0 using a signalling protocol implemented over the *Xenstore* service. After the VMs are booted and the network interfaces are configured, the functionality defined in the scenario is executed. The functionality of the scenario consists of commands that need to be run on each of the guests. These are transmitted to the guests via *Xenstore* by the central communication service. SELENA uses the *netem* qdisc [6], these qdiscs being configured on the egress queues of the guests.

B. Scaling Resources

Emulating fast networks with a high node count is challenging due to the difficulty in scaling resources. In order to overcome this, SELENA implements a time virtualization mechanism in the hypervisor to slow down the time progression on guests. In this way, the network, disk I/O rates, memory and CPU resources in guests can be scaled. The time is adjusted using a time dilation factor (TDF). In Figure 1, the hosts and the switches perceive the time adjusted by the TDF, while Dom0 and the hypervisor run in real-time. For example, if we use a TDF factor of 10, then all resources (network, disk I/O rates, CPU, memory) will appear increased tenfold within the guests, a virtual time of 1 second corresponding to 10 real time seconds. If we wish to scale only the network throughput, but keep the other resources as if we were using TDF 1 (no time dilation), then we must use certain mechanisms to scale down these other resources. In the case of CPU, we can scale down by adjusting the amount of CPU time a guest is receiving by changing the parameters of the Xen *credit2* scheduler [7]. The Xen credit scheduler exposes the *weight* and *cap* parameters for each domain, including Dom0. The *weight* parameter specifies how much CPU a domain will receive with respect to other domains on a contended host. The *cap* parameter sets the maxim amount of CPU a domain will be able to use, even if the machine running Xen has idle CPU cycles, and it is expressed as a percentage of one physical CPU. By default there is no upper cap.

Since we used compute-intensive workloads (memcached and Hadoop MapReduce, see Subsection IV-C) in our experiments, we needed to scale down the CPU that a guest can use. Taking advantage of the capping mechanism offered by the Xen credit scheduler, we set out to understand how accurate it is. We used the *Linpack* benchmark [8] as a compute-intensive

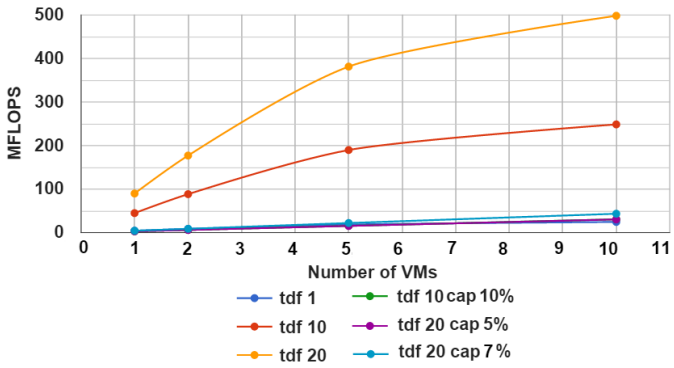

Fig. 2. VMs run simultaneously the floating point performance benchmark.

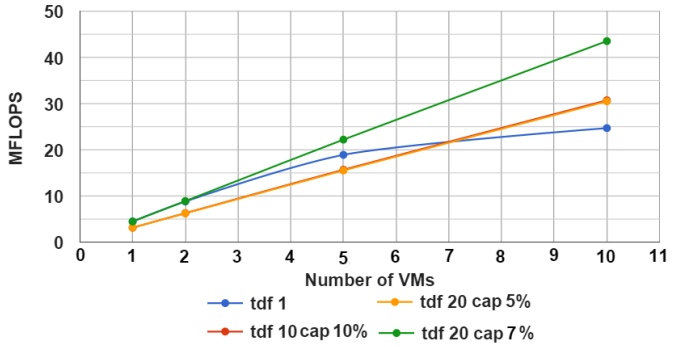

Fig. 3. VMs run simultaneously the floating point performance benchmark.

floating point performance benchmark. We ran simultaneously multiple VMs (1, 2, 5 and 10) for different TDF values (1, 10 and 20) and we report the aggregated MFLOPS value over the VMs. In this experiment, we capped the VMs' CPU time according to the formula $TDF \times cap = 100$. We notice in Figure 2 that the growth of MFLOPS is not linear with the number of VMs. This is likely to be caused by the interference between VMs running simultaneously. However, when capping the CPU time of the VMs, we obtain a linear growth for MFLOPS with the number of VMs (Figure 3), which represents the ideal aggregate MFLOPS that should have been obtained for TDF 1 had it not been for the effect of interference between VMs.

Besides CPU scaling, we also need to have the possibility to scale down the memory access speed. Since our workloads are not I/O intensive (the data used by Hadoop MapReduce is stored in RAM), we have not scaled down the disk I/O rates, but this is possible by using the *cgroups* mechanism [2].

III. The QJump Project

In this section, we explain QJump's design and expected behaviour. We chose to reproduce the network experiments presented in the QJump paper due to the fact that all the software and experiments setup are publicly available. QJump's goal is to control network interference in data centers. Network

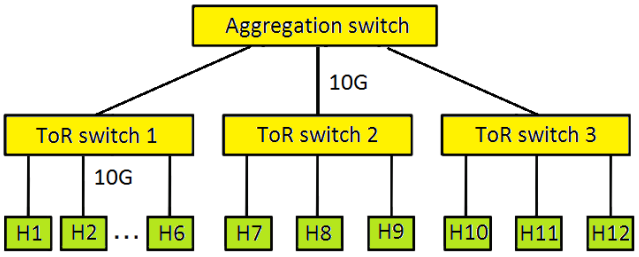

Fig. 4. Network topology for our experiments (hosts H1 - H12 and switches)

interference is defined as the delay that latency-sensitive traffic suffers from when congestion from throughput-intensive applications causes queueing in switches. QJump assigns levels (priorities) to applications depending on the type of service they require. If the application is latency sensitive, then it will require a higher level (priority) in order to have guaranteed bounded latency. On the other hand, it will be rate limited at the end host, but it will avoid queueing in switches due to its high priority. QJump thus offers the possibility of choosing between having low variance latency with low throughput or having high variance latency with high throughput for an application. QJump is implemented as a Linux Traffic Control (*tc*) module, taking advantage of the Linux queueing discipline (*qdisc*) mechanism to rate-limit packets. QJump also provides an application utility to set the priority of the application using the `SO_PRIORITY` option in the socket setup system calls. The application does not require any modifications, since the application utility is injected into unmodified executables via the Linux dynamic linker's `LD_PRELOAD` support.

The experiments carried out to show that QJump can solve network interference caused by throughput intensive applications (Hadoop) to latency-sensitive applications (PTPd, memcached) will be the focus of our reproducibility efforts.

IV. Experimental setup

We used the same topology as the authors of the QJump paper, seen in Figure 4. We have replicated their experiments on a hardware testbed and in SELENA.

A. Testbed setup

We run the applications described in the QJump paper on our testbed, in order to validate them and to have our own measurements to compare against. Our testbed consists of 12 machines, each having 6 cores Intel Xeon CPU E5-2430L v2 @ 2.40GHz with 64 GB of RAM.

B. SELENA setup

The SELENA framework was installed on a machine with 6 cores Intel Xeon CPU E5-1650 v3 @ 3.50GHz having 128GB of RAM, with hyper-threading disabled. Dom0 runs on 2 vCPUs (virtual CPUs), while the remaining 4 vCPUs are allocated to the other domains (virtual machines). We pinned the guest domains and Dom0 to vCPUs, to reduce possible artifacts that could affect our measurements due to scheduling across CPUs. Each VM that is part of the Hadoop cluster can use a maximum of 12GB of RAM, while the other VMs can use up to 3 GB of RAM.

We replicated the topology used in the QJump paper using the SELENA API. The host VMs were running the Ubuntu operating system, with the Linux kernel version 3.16. The switch VMs were running Open vSwitch [9] version 1.4.2 on top of the Debian operating system with kernel version 3.10.11. The capacity of the emulated links is 10 Gb/s with a *netem* latency of 0.02 ms.

C. Workloads

PTPd (Clock Synchronization). PTPd [1] is used to synchronize clocks in a computer network, achieving sub-microsecond accuracy on a local network. We run the PTPd master on host H8 and the client on host H1.

Memcached (Key-value Stores). Memcached is an in-memory key-value store for small chuncks of data. The memaslap [2] load generator is used to benchmark memcached by measuring the request latency. The client is running a binary protocol, with a mixed GET/SET workload of 1 KB requests in TCP mode with 2 threads and 128 concurrent requests. The memaslap client ran on the host H3, while the memcached server ran on the host H11.

Hadoop MapReduce (Data processing). Hadoop MapReduce is a system for distributed processing of large data sets. The Hadoop workload is a natural join between two uniformly randomly generated data sets. The input and output data are stored in RAM and a replication factor of 6 was used, similar to the QJump experiments. Due to memory constraints (each VM was allocated 12 of RAM), we used only half of the input data used in the QJump paper, that is 256 MB of input data. The interference caused by running Hadoop MapReduce in parallel with the other latency-sensitive applications is thus smaller in comparison to the one described in the QJump original experiments, due to the fact that a smaller quantity of data is shuffled and replicated across the machines at the end of the job. The Hadoop cluster ran on hosts H2 (master) and H4, H5, H6, H7, H9, H10, H12 (slaves).

V. Reproducing QJump experiments in Selena

The QJump module and QJump application utility can be run out of the box after being downloaded from the authors' website [3] on a testbed. However, in order to reproduce the experiments using the QJump module, we had to make several adjustments to our setup in SELENA. The Linux kernel version installed in the VMs needs to be 3.16 or newer in order to support the use of multiple transmit and receive queues per virtual interface. Prior to the Linux kernel version 3.16, the number of queues per virtual interface could not be adjusted. The maximum number of queues per virtual interface can be passed as a parameter to the Xen virtual Ethernet driver *xen-netfront* using the *max_queues* parameter. We set

[1] https://github.com/ptpd/ptpd

[2] http://libmemcached.org/libMemcached.html

the maximum number of transmit queues in each host (VM) to 8 (QJump uses 8 queues to prioritize the traffic). Since QJump uses 802.1Q VLAN priorities for prioritizing traffic, VLANs need to be configured on the hosts. We created the VLAN interfaces to have 8 transmit queues and 1 receive queue. Similarly to what was reported in [10], we had to configure the VLAN interfaces using `ifconfig` in order to have the traffic sent over them instead of the physical ones. SELENA uses the Linux *netem* qdisc [6] to support constant or stochastic latency in network links. Since QJump is implemented as a *tc* module, it was installed as a child *tc* module of the *netem* module.

The most challenging part was replicating the behaviour of the testbed switches in Open vSwitch. The authors' testbed used Arista DCS-7124fx switches and they relied on the switches' features in their experiments (8 queues per port and VLAN based prioritization). In Open vSwitch all packets are enqueued on queue 0 (the default queue). In order to support priorities, we used the PRIO qdisc, which is a queueing discipline that contains an arbitrary number of classes with different priorities. PRIO acts like a scheduler, dequeuing the classes in numerical descending order of priority. In order to enqueue the packet on the class corresponding to its VLAN priority, we use the `tc filter` extended matches `ematch` on metadata on the *vlan* attribute. We build a filter that matches on the VLAN tag taking into account the VLAN number and VLAN priority.

The Time Stamp Counter (TSC) values exposed by Xen are either native (accessed directly from the CPU register) or emulated. Since the guests in our setup are paravirtualized, the TSC is emulated [11]. The patched hypervisor used by SELENA multiplies all time sources provided to a guest with the TDF value. The QJump *tc* module relies on the TSC value for timing related decisions, checking to see if it finds an invariant TSC using the native cpuid instruction. We used the kernel clock in the QJump module instead, similarly to [10].

VI. Experimental results

We followed the description of the experiments given by Grosvenor *et al.* [1], [3]. The TDF should be chosen such that it minimizes the maximum per-CPU utilization, because the fidelity of experiments degrades if the computational resources are underprovisioned [2].

A. Comparison with testbed

PTPd (Clock synchronization). In Figure 5, we show a timeline of PTPd synchronizing a host clock in SELENA with TDF 1 and on our testbed. We notice that when running in SELENA with TDF 1, PTPd does not manage to keep the clock synchronized. We repeat the test in SELENA using different TDF values, as seen in Figure 6, and we see that as the TDF increases in value, the clock offset becomes smaller. Table I presents the values of the maximum absolute value for the PTPd clock offset for the timelines from Figures 5 (testbed) and 6 (we did not take into account the synchronization period at the beginning of the interval). We can clearly see that, amongst the TDF values that we tried, TDF 50 provides the best fidelity for this experiment in comparison with the testbed value, while the values for TDF 10 and 20 are similar. If we were to use a TDF of 50 in our experiments, our experiments will take 50 times more time. Hence, we chose a TDF of maximum 20 for the subsequent experiments in order to have an acceptable running time, but with good experimental fidelity.

We next present timelines of PTPd synchronizing a host clock while simultaneously running the Hadoop MapReduce job in SELENA and on our testbed. With TDF 1, the clock offset reaches values of 70000μs in SELENA, being higly

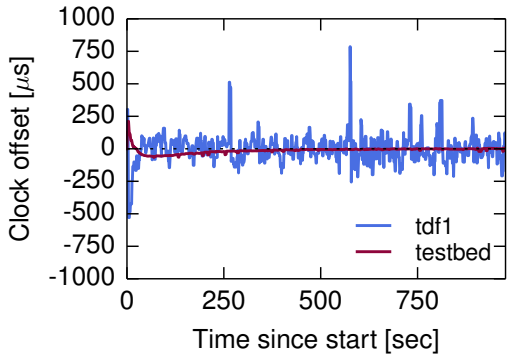

Fig. 5. PTPd in SELENA with TDF 1 and on the testbed.

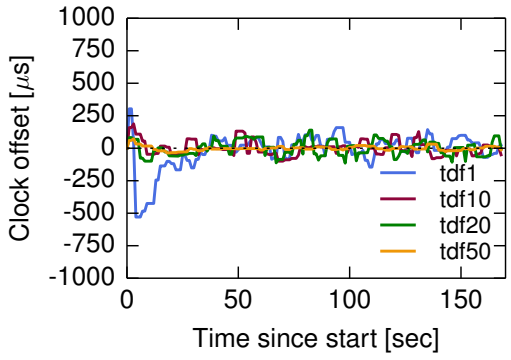

Fig. 6. PTPd in SELENA with different TDF values.

TABLE I
MAXIMUM ABSOLUTE VALUE FOR PTPD CLOCK OFFSET.

Experiment	**Platform**	Maximum clock offset (μs)
PTPd	Testbed	20.626
PTPd	SELENA TDF=1	158.547
PTPd	SELENA TDF=10	131.354
PTPd	SELENA TDF=20	139.284
PTPd	SELENA TDF=50	32.033
PTPd + Hadoop	Testbed	1240.142
PTPd + Hadoop	SELENA TDF=10	31045.623
PTPd + Hadoop	SELENA TDF=20	1406.18

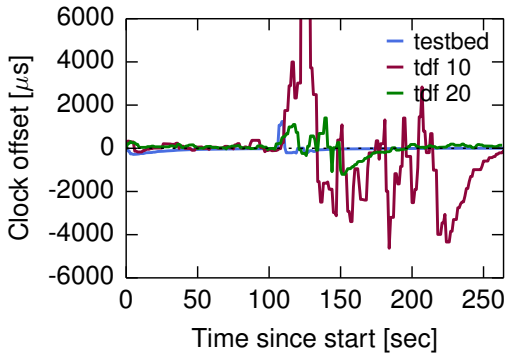

Fig. 7. PTPd and Hadoop MapReduce in SELENA with TDF 10 and TDF 20 and on the testbed.

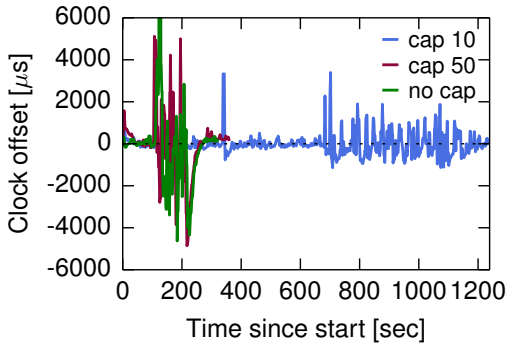

Fig. 8. PTPd and Hadoop MapReduce in SELENA with TDF 10 using different cap values

TABLE II
COMPARISON OF REQUEST LATENCIES STATISTICS.

Experiment	Platform	Mean	Std
Memcached	Testbed	329.42	62.05
Memcached	SELENA TDF=10	435.34	142.52
Memcached+Hadoop	Testbed	369.09	158.08
Memcached+Hadoop	SELENA TDF=10	2733.97	3303.93

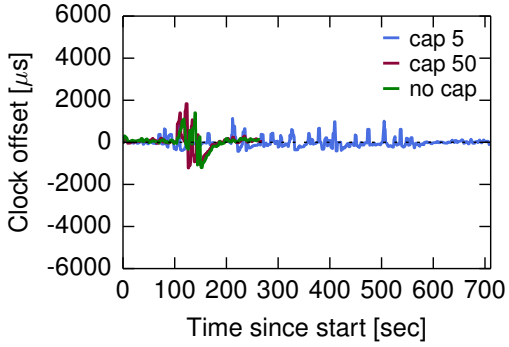

Fig. 9. PTPd and Hadoop MapReduce in SELENA with TDF 20 using different cap values

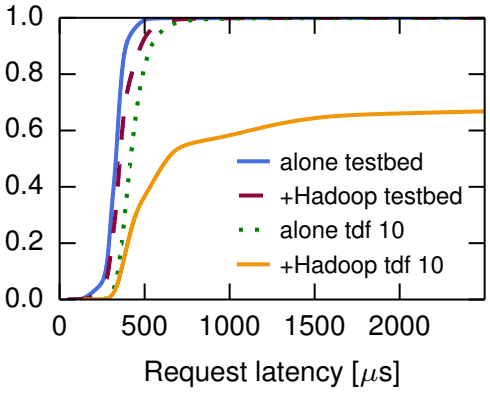

Fig. 10. memcached and Hadoop MapReduce on the testbed and on SELENA with TDF 10

unrealistic, while on the testbed the maximum clock offset is 1240.142μs. We see from Figure 7 and Table I that running PTPd and the Hadoop MapReduce job in SELENA with TDF 20 produces the best fidelity compared to the testbed results. If we were to use a higher TDF, we would obtain better experimental fidelity. We apply the capping mechanism described in Section II to the VMs running the Hadoop job in order to limit the computational resources that the VM perceives as an effect of the higher TDF. Figure 8 presents a timeline of PTPd and a Hadoop MapReduce job running simultaneously in SELENA with TDF 10 and using different values for the *cap* parameter (5, 50 and no cap). We observe that when limiting the CPU time to 10% on each VM, the runtime of the Hadoop MapReduce job increased in comparison with the runs in which we do not use a cap or we use a cap of 50%, but the interference caused by Hadoop MapReduce is smaller in amplitude. We notice a similar trend for TDF 20 in Figure 9.

As the TDF increases, the fidelity of the results obtained in the emulation framework increases. When running PTPd in SELENA with a high TDF value, the behaviour of the application is very close to the one observed on the testbed. However, in the case of compute or I/O intensive applications, like Hadoop, the behaviour of the application changes due to the increased resources it has access to, making it hard to replicate its exact behaviour.

Memcached (Key-Value Stores). Figure 10 shows the distribution (CDF) of memcached request latencies when running on an idle network and on a shared network with Hadoop MapReduce. Table II presents the mean and standard deviation of request latencies for each experiment. When running memcached on an idle network, the mean results on the testbed and in SELENA with TDF 10 differ by approximately 100μs. This difference is probably caused by the fact that the switch in the testbed is an Arista switch, while we use Open vSwitch. To mitigate the differences between the testbed switches and the ones used in emulation, SELENA offers the possibility of integrating a custom switch model. We leave to future work the integration of a custom switch in our experiments.

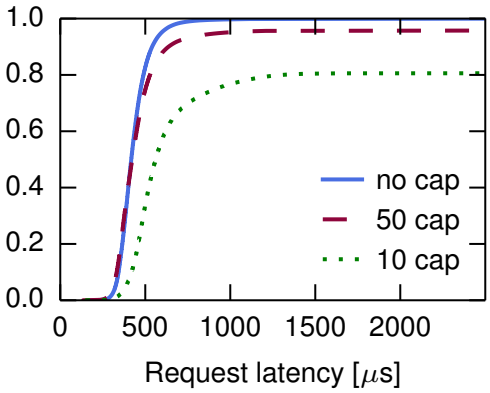

Fig. 11. memcached on SELENA with TDF 10 and different cap values

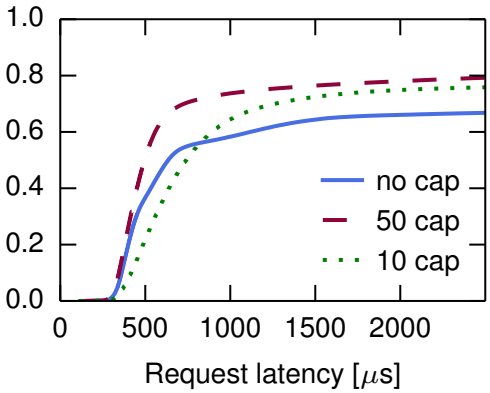

Fig. 12. memcached and Hadoop MapReduce on SELENA with TDF 10 and different cap values

When running Memcached on a shared network with Hadoop MapReduce there is a significant difference between the results on the testbed and in SELENA, showing that a higher TDF is needed when running complex applications like Hadoop MapReduce. Also, the CPU is increased 10 times and this leads to an increased number of memcached requests being sent and to the Hadoop MapReduce job finishing quicker. In order to mitigate this, we capped the CPU time per VM using the *cap* parameter (see Figure 11 and Figure 12). Although this lowered the number of requests sent by the memaslap client, part of the requests started having high latency, even when memcached was running on an idle network, because the requests could not be processed timely by memcached given the reduced CPU time. This experiments show that compute-intensive or I/O intensive applications are hard to emulate due to differences between running the applications on the testbed (which has multi-core machines) and in SELENA (VMs use a single core) and due to the effect of time dilation.

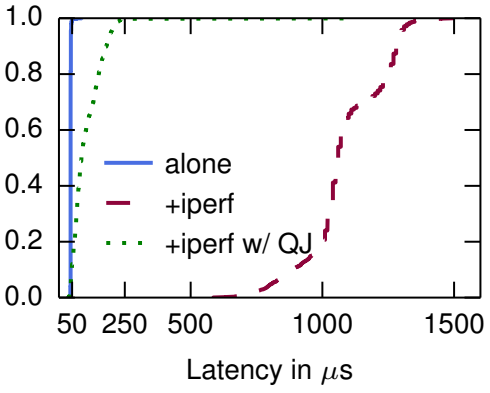

Fig. 13. ping and iperf with and without Qjump in SELENA using TDF 20

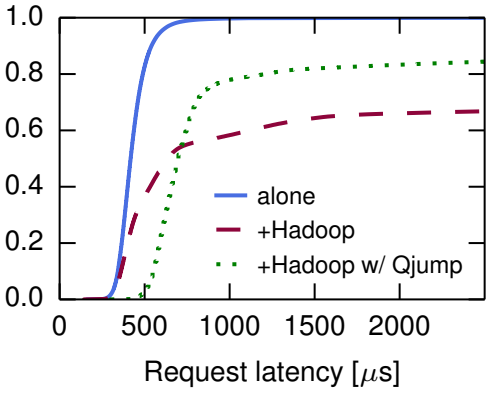

Fig. 14. memcached and Hadoop with QJump in SELENA using TDF 10

B. QJump reduces network interference

The first experiment that uses QJump to resolve network interference is running simultaneously *ping* (which emulates low latency RPCs (Remote Procedure Calls)) and *iperf* (which emulates bulk transfer). *ping* runs between H3 and H11 and *iperf* runs between H1 and H12, and between H8 and H11 (2 distinct streams). We set *ping* to the highest QJump level 7. The authors measured the packet latency across a switch, while we measured the packet latency reported by ping. The packets' latency is reduced by over 100 times when using QJump (Figure 13). We obtain similar results to the original QJump ones, showing that prioritization inside of switches does not resolve all interference. The median *ping* latency in SELENA is 44.9 μs and it increases to 1060 μs when *iperf* is running alongside *ping*. But with QJump enabled, the median latency decreases to 86.2 μs. From this experiment and the PTPd ones, we can see that the behaviour of applications like PTPd and *ping*, which are network-centered, can be successfully emulated.

The second experiment aims to resolve network interference experienced by memcached when it shares the network with Hadoop. In this experiment, memcached is configured at an

intermediate QJump level (level 4) and rate-limited to 5Gb/s, as explained in [3]. Figure 14 shows the distribution of memcached request latencies when running on an idle network, a shared network and a shared network with QJump enabled. In this experiment, QJump reduces the network interference, but it does not solve it completely. Using a lower rate limit for memcached in the QJump configuration reduces even further the network interference.

VII. Related work

A. Reproducibility

Our study is an in-depth performance analysis of the SELENA network emulation framework while concomitantly reproducing experiments described in the NSDI 2015 paper [1]. Several other papers described the experience of reproducing experiments from previously published papers. In [12], the authors described the experiments from 3 published networking research papers they had reproduced using Mininet Hi-Fi, a container-based emulation framework [12]. Additionally, the authors presented the experience of students attending a networking class, whose final project was reproducing networking research. The students managed to reproduce results from 16 other published networking research papers in Mininet Hi-Fi and documented their work on the class blog. However, the link speed in some experiments was scaled down to 100 MB/s from 1GB/s or 10GB/s, due to Mininet's difficulty to emulate high throughput links, but the results reproduced are quantitatively equivalent to the results in the papers. The experiment code of the 3 projects reproduced was obtained directly from the original authors. Howard *et al.* [13] repeated the performance analysis of the Raft consensus protocol. They developed a new implementation of the protocol and an event-driven simulation framework. They managed to reproduce the original results and they also proposed optimizations to the protocol, acknowledging the instrumental help of the authors of the Raft protocol in their reproducibility efforts. Another reproducibility study centers around repeating the performance analysis of Xen [14]. The authors were successful in reproducing the results and they also extended the analysis with additional tests.

Several studies focus only on determining whether the information needed to reproduce the results presented in a research paper is available, without actually repeating all the experiments associated with the papers and comparing their results with the original results. Kurkowski *et al.* [15] surveyed the 2000-2005 proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc) (114 papers that used simulation) and found that less than 15% of them are repeatable. Kovacevic [16] analyzed 15 papers published in the IEEE Transactions on Image Processing and contended that for none of the papers the authors had provided the source code of the algorithms described, while for 33% of them the input data is available. A more detailed study [17] of 134 papers published in the same journal in 2004 concludes that only 9% of the papers have code available online, while one third have input data available. The SIGMOD conference set up an experimental reproducibility initiative in 2008 which has been running ever since [18]. A methodology for reproducibility is presented in [19]. For 21 papers accepted at the Euro-Par 2013, a reproducibility score is computed based on the criteria defined in the methodology. More recently, Collberg *et al.* [20] studied 613 papers from top systems conferences and journals to see if they can build and execute the source code of the projects. They were successful in less than 25% of the cases.

B. Network Simulation and Emulation Tools

Network simulation or emulation frameworks represent a powerful tool for researchers. In the context of reproducibility, these frameworks offer the possibility of setting up experimental environments similar to testbeds on which the experiments one wants to reproduce were run, but without using additional hardware and without incurring the overhead of building the testbed. However, the experimental results obtained on such frameworks are not always realistic due to different factors.

Simulation based frameworks are often employed by researchers in order to evaluate their prototypes. The most popular simulators being ns-2 [21], ns-3 [22] and OMNET++ [23]. Simulators employ an event-driven simulation clock and simplified models for hardware and network protocols. Simulations are usually lengthy in time, while their experimental fidelity may not be ideal due to the network models used.

Emulation brings more realism in the experimental results by allowing the use of unmodified applications and operating systems. DieCast [24] is an emulator that employs full-system virtualization and it is based on Xen. It uses time dilation [25] to scale the resources. DieCast offers mechanisms for scaling down the disk I/O rates and CPU in order to support disk intensive applications. However, DieCast requires guest modifications and its patches for time dilation are outdated and do not work with newer Xen versions. It also does not offer an API to construct scenarios, making it difficult to use as a tool for reproducing experiments. Mininet [12] is a popular emulation framework. It uses lightweight virtualization based on Linux Containers. Unfortunately, as reported in [2], Mininet fails to provide experimental fidelity for scenarios with high-throughput links or larger networks. VT-Mininet [26] enhances Mininet with a time dilation mechanism similar to TimeKeeper [27], modifying the Linux kernel time-related system calls. VT-Mininet provides virtual time per container or a global virtual time and has a TDF adaptor that dynamically adjusts the TDF based on the CPU utilization to ensure faster experiment completion while maintaining experimental fidelity. We chose SELENA as our framework for reproducing network experiments due to its high experimental fidelity and because the project is open source.

VIII. Conclusion

In this paper, we explore the tradeoff between experimental time and fidelity of reproduction in SELENA [2] by reproducing network experiments presented in QJump [1]. While we were successful in reproducing the experiments with a degree

of error, we realized that SELENA's emulation capabilities are limited for measurements with microsecond granularity and with respect to certain applications. As presented in [2], SELENA manages to successfully emulate high throughput links and millisecond granularity accuracy for measurements, but our measurements require microsecond granularity. Compute-intensive and I/O intensive applications' behaviour is difficult to replicate in a VM which runs on a single core, while on the testbed these applications were run on multi-core machines. Also, because time dilation scales all the resources, applications like Hadoop are hard to emulate. On the other hand, network-centered applications, like PTPd, ping and to a certain extent memcached (if the workload used is network bound), can be successfully emulated.

ACKNOWLEDGMENTS

We thank the reviewers and our shepherd, Marco Mellia, for valuable comments regarding the paper. The authors would like to thank Dimosthenis Pediaditakis, Matthew Grosvenor and Charalampos Rotsos for their assistance during the course of this project. Diana Andreea Popescu is funded by the EU FP7 Marie Curie ITN METRICS (grant agreement no. 607728).

REFERENCES

[1] Grosvenor, Matthew P. and Schwarzkopf, Malte and Gog, Ionel and Watson, Robert N. M. and Moore, Andrew W. and Hand, Steven and Crowcroft, Jon, "Queues Don't Matter when You Can JUMP Them!" in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'15. Berkeley, CA, USA: USENIX Association, 2015, pp. 1–14. [Online]. Available: http://dl.acm.org/citation.cfm?id=2789770.2789771

[2] 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. [Online]. Available: http://doi.acm.org/10.1145/2658260.2658274

[3] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. M. Watson, A. W. Moore, S. Hand, and J. Crowcroft, "Jump The Queue," http://www.cl.cam.ac.uk/research/srg/netos/qjump/, 2015, online; accessed 13 December 2015.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03. New York, NY, USA: ACM, 2003, pp. 164–177. [Online]. Available: http://doi.acm.org/10.1145/945445.945462

[5] Xen, "Xen Networking," http://wiki.xenproject.org/wiki/Xen_Networking, online; accessed 14 December 2015.

[6] S. Hemminger, "Network emulation with NetEm," 2005.

[7] Xen , "Xen Credit Scheduler," http://wiki.xen.org/wiki/Credit_Scheduler, online; accessed 14 December 2015.

[8] Linpack , "The LINPACK Benchmark," http://people.sc.fsu.edu/~jburkardt/c_src/linpack_bench/linpack_bench.html, online; accessed 15 December 2015.

[9] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The Design and Implementation of Open vSwitch," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'15. Berkeley, CA, USA: USENIX Association, 2015, pp. 117–130. [Online]. Available: http://dl.acm.org/citation.cfm?id=2789770.2789779

[10] A. Klimovic and C.-Z. Lee, "CS 244 '15: QJUMP-DELAY GUARANTEES IN DATACENTER NETWORKS," https://reproducingnetworkresearch.wordpress.com/2015/05/31/cs-244-15-qjump-delay-guarantees-in-datacenter-networks/, 2015, online; accessed 12 December 2015.

[11] Xen, "Xen," http://xenbits.xen.org/docs/4.3-testing/misc/tscmode.txt, online; accessed 15 December 2015.

[12] 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. [Online]. Available: http://doi.acm.org/10.1145/2413176.2413206

[13] H. Howard, M. Schwarzkopf, A. Madhavapeddy, and J. Crowcroft, "Raft refloated: Do we have consensus?" *SIGOPS Oper. Syst. Rev.*, vol. 49, no. 1, pp. 12–21, Jan. 2015. [Online]. Available: http://doi.acm.org/10.1145/2723872.2723876

[14] B. Clark, T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne, and J. N. Matthews, "Xen and the art of repeated research," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ser. ATEC '04. Berkeley, CA, USA: USENIX Association, 2004, pp. 47–47. [Online]. Available: http://dl.acm.org/citation.cfm?id=1247415.1247462

[15] S. Kurkowski, T. Camp, and M. Colagrosso, "Manet simulation studies: The incredibles," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 9, no. 4, pp. 50–61, Oct. 2005. [Online]. Available: http://doi.acm.org/10.1145/1096166.1096174

[16] J. Kovacevic, "How to encourage and publish reproducible research," in *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, vol. 4, April 2007, pp. IV–1273–IV–1276.

[17] P. Vandewalle, J. Kovacevic, and M. Vetterli, "Reproducible research in signal processing," *Signal Processing Magazine, IEEE*, vol. 26, no. 3, pp. 37–47, May 2009.

[18] I. Manolescu, L. Afanasiev, A. Arion, J. Dittrich, S. Manegold, N. Polyzotis, K. Schnaitter, P. Senellart, S. Zoupanos, and D. Shasha, "The repeatability experiment of sigmod 2008," *SIGMOD Rec.*, vol. 37, no. 1, pp. 39–45, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1374780.1374791

[19] A. Carpen-Amarie, A. Rougier, and F. Lubbe, "Stepping stones to reproducible research: A study of current practices in parallel computing," in *Euro-Par 2014: Parallel Processing Workshops*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2014, vol. 8805, pp. 499–510. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-14325-5_43

[20] C. Collberg, T. Proebsting, G. Moraila, A. Shankaran, Z. Shi, and A. Warren, "Measuring reproducibility in computer systems," University of Arizona, Tech. Rep., 2014.

[21] T. Issariyakul and E. Hossain, *Introduction to Network Simulator NS2*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[22] T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley, "Ns-3 project goals," in *Proceeding from the 2006 Workshop on Ns-2: The IP Network Simulator*, ser. WNS2 '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1190455.1190468

[23] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, ser. Simutools '08. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 60:1–60:10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1416222.1416290

[24] D. Gupta, K. V. Vishwanath, M. McNett, A. Vahdat, K. Yocum, A. Snoeren, and G. M. Voelker, "Diecast: Testing distributed systems with an accurate scale model," *ACM Trans. Comput. Syst.*, vol. 29, no. 2, pp. 4:1–4:48, May 2011. [Online]. Available: http://doi.acm.org/10.1145/1963559.1963560

[25] D. Gupta, K. Yocum, M. McNett, A. C. Snoeren, A. Vahdat, and G. M. Voelker, "To infinity and beyond: Time-warped network emulation," in *Proceedings of the 3rd Conference on Networked Systems Design & Implementation - Volume 3*, ser. NSDI'06. Berkeley, CA, USA: USENIX Association, 2006, pp. 7–7. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267680.1267687

[26] 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. [Online]. Available: http://doi.acm.org/10.1145/2774993.2775012

[27] J. Lamps, D. M. Nicol, and M. Caesar, "Timekeeper: A lightweight virtual time system for linux," in *Proceedings of the 2Nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, ser. SIGSIM PADS '14. New York, NY, USA: ACM, 2014, pp. 179–186. [Online]. Available: http://doi.acm.org/10.1145/2601381.2601395