Faithful Reproduction of Network Experiments

Dimosthenis Pediaditakis Charalampos Rotsos Andrew W. Moore
Computer Laboratory, University of Cambridge
{firstname.lastname}@cl.cam.ac.uk

ABSTRACT

The proliferation of cloud computing has compelled the research community to rethink fundamental aspects of network systems and architectures. However, the tools commonly used to evaluate new ideas have not kept abreast of the latest developments. Common simulation and emulation frameworks fail to provide scalability, fidelity, reproducibility and execute unmodified code, all at the same time.

We present SELENA, a Xen-based network emulation framework that offers fully reproducible experiments via its automation interface and supports the use of unmodified guest operating systems. This allows out-of-the-box compatibility with common applications and OS components, such as network stacks and filesystems. In order to faithfully emulate faster and larger networks, SELENA adopts the technique of *time-dilation* and transparently slows down the passage of time for guest operating systems. This technique effectively virtualizes the availability of host's hardware resources and allows the replication of scenarios with increased I/O and computational demands. Users can directly control the trade-off between fidelity and running-times via intuitive tuning knobs. We evaluate the ability of SELENA to faithfully replicate the behavior of real systems and compare it against existing popular experimentation platforms. Our results suggest that SELENA can accurately model networks with aggregate link speeds of 44 Gbps or more, while improving by four times the execution time in comparison to ns3 and exhibits near-linear scaling properties.

Categories and Subject Descriptors

C.2.5 [**Computer Systems Organization**]: Computer - Communication NetworksLocal and Wide-Area Networks; D.4.8 [**Software**]: Operating SystemsPerformance

Keywords

Network experimentation, emulation, Virtualisation

1. INTRODUCTION

The adoption of networking technologies by a widening spectrum of applications and environments has highlighted a series of limitations in the design of the predominant protocols and architectures. In order to support the growing requirement for scalable network performance, an equal effort is put towards the development of network experimentation tools.

We identify three key properties in the realm of network modeling and experimentation: fidelity, reproducibility and scalability. *Fidelity* characterizes the ability of the experiment to replicate specific system behavior with high precision and accuracy. Examples of network experimentation fidelity can be characterized by the requirement for functional realism in terms of network hosts and network topology, the ability to reuse realistic applications and traffic models and the accurate recreation of the timing properties of the real system.

Scalability is an equally important property for experimentation platforms, defined by the ability to support and gracefully manage network experiments of growing size. This requirement is extremely difficult to meet with respect to current network architectures, primarily due to the exponential increase in their size, link speed and complexity. Experimental scalability can be further decomposed in three functional aspects: *execution time scalability*, which describes the required wall-clock time to replicate an experiment, *resource scalability*, which characterizes the ability of the platform to minimize hardware requirements, and *fidelity at scale*, which reflects the ability to maintain high fidelity as the size of the experiment increases. The three scalability aspects exhibit *Pareto efficiency*: a platform cannot improve one of them without affecting negatively the remaining two aspects.

Reproducibility is a third key property for network experimentation which has gained significant interest in the recent years [13, 32]. Reproducibility describes the ability of the platform to export and replicate experimental scenarios and its results. It requires a rich automation abstraction for the configuration of all experimental details, but must also provide guarantees on fidelity across heterogeneous hardware platforms (e.g. control the impact of host's available processing capacity on the results obtained from an experiment execution).

This paper presents SELENA, an holistic network experimentation tool supporting reproducibility and exposing explicit control on the trade-off between scalability and fidelity. The tool runs *in-a-box* and can faithfully emulate complex network architectures with high-speed links. We argue that emulation provides the optimal approach to maximize the fidelity of an experiment, due to its support for unmodified real-world software and OS. The design of SELENA combines a variety of network experimentation optimizations (e.g. VM-based emulation, link modeling, time dilation, automation API etc.) under a common framework. Unlike the pop-

Tool Name	Readily available	Reproducible Experiments	Hardware requirements	Real Stacks	Unmodified applications	SDN support	Fidelity at scale		Execution Speed
							Node	Link speed	
	Simulation								
NS2 [16]	✓	✓	Low	×	×	×	●●●	●●○	●○○
NS3 [15]/ OMNeT++ [34]	✓	✓	Low	×	×	partially	●●○	●○○	●○○
FS-SDN [12]	✓	✓	Low	×	×	✓	●●●	●●○	●●○
NS Cradle [17] / DCE [32]	✓	✓	Low	✓	no full POSIX	partially	●●○	●○○	●○○
	Simulation-Emulation Hybrid								
OpenVZ-S3F [37]	✓(outdated)	×	Low	✓	✓	✓	●○○	●○○	●○○
SliceTime [35]	✓(outdated)	×	Low	✓	✓	×	●○○	●○○	●○○
	Emulation								
ModelNet [33]	✓(outdated)	×	Average	✓	✓	×	●●○	●○○	●●● (RT)
DummyNet [5]	✓	×	Average	✓	✓	×	●●○	●○○	●●● (RT)
MiniNet [13]	✓	✓	Low	✓	✓	✓	●●○	●○○	●●● (RT)
	Time-controlled Emulation								
DieCast [10]	×	×	Low	✓	✓	×	●●○	●●●	●●○
TimeJails (TVEE) [8]	×	×	Low	✓	✓	×	●●○	●●○	●●○
SELENA (this work)	✓	✓	Low	✓	✓	✓	●●○	●●●	●●○
	Testbeds								
Planetlab [26]	✓	n/a	High	✓	✓	custom OVS	●●○	●○○	●●● (RT)
Ofelia [21]	✓	n/a	High	✓	✓	✓	●●○	●○○	●●● (RT)

Table 1: Comparison of popular network experimentation platforms across different dimensions

ular container-based Mininet platform [13], SELENA employs OS virtualization achieving better resource control and isolation and wider support for network stacks and OSes [36].

In order to overcome the inherent scalability limitations of network emulation, SELENA revisits the idea of *Time Dilation* [11]: controlling resource availability per unit of time, by virtualizing time progression in the guests of the experiment. This approach fundamentally provides users with direct control on the scalability trade-offs of an experiment. For example, time dilation opens an opportunity for near-real time execution of a large number of low-fidelity experiments to explore a variable-space, whereas when precise behavior details are sought, an experimenter can run a higher-fidelity experiment (or set of experiments) by increasing respectively the time dilation factor (TDF). Our approach improves earlier approaches in time-controlled emulation in two ways: first, it provides support for recent Xen versions and requires zero guest modifications (if it works on Xen, it works on SELENA) and second, it offers better experimental automation and reproducibility. In addition, due to the varying performance profiles of existing SDN forwarding devices [30], SELENA supports high precision switch emulation models, which further improve the fidelity of OpenFlow-based experiments.

In summary, SELENA provides the following key features:

- **Reproducibility**: A Python API can be used by experimenters to describe custom experiments, automating their creation, deployment and execution.
- **Fidelity**: Time-dilation functionality [11] scales guests' view of physical resources helping to accurately model complex networks. We successfully patched the latest version of Xen and tested it against recent Linux and FreeBSD kernels. SELENA supports any paravirtualized (PV) guest, requires zero modifications and provides full POSIX compatibility. Real-world applications and network stacks can be directly tested on our platform.
- **Scalability**: Slowing down the progression of time at guests enables support for high link speeds (40 Gbps or higher) and larger networks, within a single host. In section 4.4 we describe a methodology for exploring the scalability and accuracy limits that a given TDF value can provide, helping users to better understand how to efficiently use SELENA.
- **Software Defined Networking (SDN)**: Apart from supporting all popular software switch and controller implementations, we also incorporate a flexible OpenFlow switch emulation model which offers better realism in terms of control plane behavior.

To the best of our knowledge, SELENA is the only open-source network emulation tool [1] which provides all the above features. The rest of this paper is organized as follows. Section 2 provides an overview of related network experimentation platforms. Section 3 describes the architecture and implementation details of SELENA. Section 4 evaluates the fidelity and scalability of SELENA against a real-world deployment, and compare it against the Mininet emulation framework and the ns-3 simulation platform. We also evaluate the accuracy of our switch model and the ability of SELENA to replicate real network applications. Finally, we discuss the limitations of our work in Section 5, point out future work directions and conclude in Section 6.

2. BACKGROUND AND RELATED WORK

This section presents a survey of network experimentation practices, focusing on aspects of fidelity, reproducibility and scalability. We categorize relevant approaches into four broad categories.

Simulation is a popular experimentation method, employing simplified models to replicate network functionality. ns-2 [16], ns-3 [15] and OMNET++ [34] are some of the most popular event-driven network simulators, supporting a plethora of simulation models (e.g. for link properties, network protocols etc.). These platforms aim to provide good resource scalability, deterministic reproducibility and simplify the early-stage evaluation of new ideas by masking the complexity of system-level details. However, their fidelity is typically sacrificed in favor of scalability, influenced by the simplistic assumptions of the network and system models.

Many users consider simulation a weak method for testing new ideas because it does not incorporate real-world network stacks and it does not support execution of unmodified applications. This weakness has motivated recent efforts like DCE [32] and NSC [18], which attempt to extend ns-3 and bridge the API mismatch between real-world applications and simulation. While these attempts are a big leap forward, they compromise scalability (the main advantage of simulation) and offer partial POSIX compatibility. Simulation platforms have also added support for recent approaches to network control. For example, FS-SDN [12] seeks to help studying the macroscopic impact of novel network control applications on data-plane performance using lightweight flow-level models.

Emulation aims to improve experimentation fidelity by enabling the reuse of real world protocols and applications within a vir-

[1] Available at http://selena-project.github.io

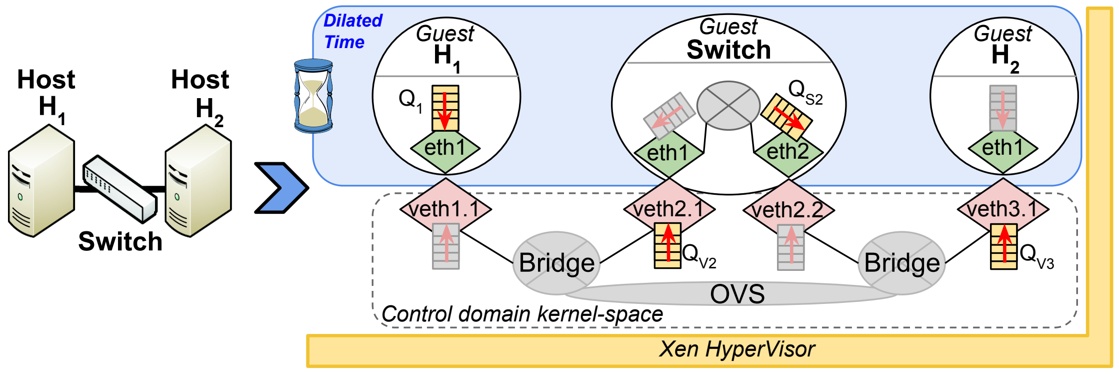


Figure 1: SELENA emulation architecture builds on Xen. Emulated hosts run in dilated time, control-domain perceives real time.

tual network. The majority of relevant platforms replace the network layers below the data-link with simplified models which replicate their functionality. The ModelNet [33] project established a pioneering approach in scalable emulation of Internet topologies. Its architecture fuses edge hosts running unmodified applications, with nodes emulating virtual network topologies using DummyNet [5]. ModelNet improved scalability by increasing hardware requirements and could only parallelize execution at the extent which a particular application and topology allows it.

Recent efforts in emulation frameworks aim to scale-down resource requirements and fit large network experiments in a single host, without significant precision losses. The developments in computer resource virtualization allowed emulation platforms to isolate instances of virtual network components and improved control on resources usage [4, 27]. *Mininet* [13] is the most popular framework of this category, highlighting a reproducibility requirement in network experimentation. Mininet automates the experimentation workflow, so that a particular high-level scenario description consistently recreates the same network topologies, configurations, traffic patterns and application behaviors across different host architectures. In order to improve further the fidelity in replicating SDN architectures, Gupta *el al.* [12] developed a control plane proxy service for Mininet, enabling users to integrate switch models into network experiment.

Nonetheless, the increasing complexity of emulated systems affects their computational scalability and restricts the support for large network sizes and link speeds in real time. This limitation has precipitated the development of **time-controlled emulation**, where experiment time is virtualized. By executing scenarios at a slower pace, available hardware resources are scaled and therefore the accuracy of results is improved. Of particular note is *DieCast* [11] which, like SELENA, employs full-system emulation using Xen to scale CPU, disk and network IO resources by dilating the time-progression in guests. Unfortunately, DieCast's patches for time dilation are no longer compatible with modern versions of Xen, it requires guest modifications, it does not offer automation mechanisms for reproducible experimentation and, finally, it does not support realistic OpenFlow switch models. Similarly, TimeJails (TVEE) [8] is a VM-based solution with time virtualization support which combines multiple virtual routing instances in a single virtual machine to improve resource scalability. TVEE applies dynamic hardware allocation to maximize utilization and allows emulating large numbers of virtual nodes per host.

Testbed infrastructures like Ofelia [21] and Planetlab [26], are large virtualized computer infrastructures, used for the deployment of experiments. Such platforms achieve high fidelity, similar to multi-host emulation at the cost of higher hardware requirements. Kim *el al.* [19] highlighted potential measurement noise and biases in network measurements on Planetlab nodes, due to the shared nature of network testbeds and the limited control on resources.

Hybrid approaches like OpenVZ-based S3F [37] and SliceTime [35] provide an alternative towards experimental resource scalability and fidelity, by combining the emulation with simulation. Specifically, they typically enforce weak virtual time synchronization between emulated nodes, using a centralized simulation engine. SliceTime, for example, uses ns-3 to coordinate the execution of Xen guests by controlling the hypervisor's scheduling policy. Guests run independently for a quantum of time and a global service synchronizes them periodically with the ns-3 clock.

Table 1 presents a comparison of the presented network experimentation frameworks. Each entry categorizes a platform with respect to its fidelity (Real Stacks, Unmodified applications, SDN support), scalability (Hardware requirements, Fidelity at scale, Execution Speed) and reproducibility. SELENA falls under the category of "*time-controlled emulation*" and aims to combine the competitive advantages of Mininet and DieCast. It supports reproducible experimentation, it presents good scalability properties and furthermore, its functionality can be extended with custom OpenFlow switch models.

3. SELENA DESIGN

This section presents the design of SELENA. Using as reference a simple experiment scenario, we provide a high level description of its execution model (§ 3.1), followed by a presentation on how a user can construct experiments over the platform via its automation API (§ 3.2). Furthermore, we elaborate on the details of the underlying mechanisms which improves resource scalability (§ 3.3), data plane fidelity (§ 3.4), and control plane realism (§ 3.5).

3.1 Overall architecture

SELENA revisits the idea of network emulation using OS-level virtualization. Recent efforts in network experimentation [13] use lightweight approaches, like container-based virtualization. We argue that hypervisor-based virtualisation provides better *heterogeneity support* (e.g. supports a wide OS range[2]) and improved *resource control granularity* [36]. SELENA uses Xen [3], a mature open-source hypervisor which exhibits good performance and scalability properties [22]. Furthermore, the Xen Cloud Platform (XCP) provides a rich API (xen-api), supporting remote Virtual Machine

[2]http://wiki.xen.org/wiki/DomU_Support_for_Xen

(VM) configuration and resource control. Using *xen-api*, SELENA automates the deployment and execution of virtual network topologies and experimental scenarios.

Figure 1 presents a trivial network experiment topology; two hosts interconnected through a switch. SELENA uses a simple mapping mechanism: each host or network device is mapped to a VM, while each link maps to a pair of virtual network interfaces (one for each guest) bridged in the Dom0. With respect to Figure 1, the hosts (H_1 and H_2) and the switch (SW) are mapped to separate VMs. Host SW is configured with two network interfaces, each bridged in the Dom0 using a separate bridge with an interface from hosts H_1 and H_2. The interfaces illustrated with pink color (e.g. $veth1.1$) are Xen-specific *netback* devices, transparent to the guests of the experiment. The resulting virtual machine configuration is a precise representation of the emulated network scenario, replicating both topological and functional properties, like queue sizes, link speeds and host resource limitations. SELENA also virtualizes time for all participating guests (blue sandbox), allowing experimenters to control the trade-off between resource and time scalability and fidelity.

3.2 Reproducible experiments

SELENA exposes a programming API to specify, deploy and execute network experiments. We share the same views with the authors of Mininet [13] on experimental reproducibility and automation. We believe that both requirements are fundamental for a network experimentation platform, improving repeatability of results and simplifying the frequent trial-and-error cycles of applied research. The mechanism outlined in this section makes experimentation with SELENA simple and user-friendly. Experimenters can effortlessly re-run experiments with varying parameters, like queue sizes, link speeds, topologies and application or even host configurations. An experiment can be repeated on different platforms and hosts, requiring only the installation of SELENA and the script defining the network topology and functionality of the experiment. The process of defining and executing a network experiment in SELENA consists of four steps.

Step 1 - *Network description*

SELENA provides a Python API for the definition of the network hosts and their topology, as well as their configuration parameters. Listing 1 presents the topology definition of the experiment in Figure 1. A network definition is typically composed by the user and contains two types of objects: nodes and links. A node object contains several fields defining Xen-specific guest resource configurations (e.g. memory size, virtual CPUs and affinity), the template of the VM and the node network interfaces along with their initial configuration (e.g. IP and MAC address, queue sizes). The link object is simpler and specifies the link end-points (node and interface IDs) along with the link capacity and latency characteristics.

Step 2 - *Experiment deployment*

During deployment, SELENA parses the network description, checks for common syntax or semantic errors (e.g. malformed IP addresses, duplicate links, platform resources availability) and invokes the relevant *xen-api* callbacks to create the VMs of the experiment and then installs the required network bridges in Dom0 to replicate the topology.

Step 3 - *Experiment initialization*

SELENA uses a distributed control and monitor framework to synchronize host configuration and scenario execution, which consists of a low overhead daemon running on each host and a central coordination service running on Dom0. The guest daemon is responsible to report host status information to the coordination service and to execute the commands of the experiment in a timely manner. The daemon communicates with the coordination service using a signaling protocol over the Xenstore service, thus minimizing the interference with the emulated network's resource. During initialisation, SELENA boots sequentially all the VMs which have been created in the previous step and configures the network interfaces using the coordination service.

```
# Node-0 (H1)
newNode(
  0, "H1"                # unique ID, name label
  NodeType.LINUX_HOST,   # guest template
  [ (1000,               # 1st interface, queue len
    "RANDOM",            # MAC address
    "10.0.1.2",          # IP address
    "255.255.255.0",     # NetMask
    "10.0.1.1"),         # Gateway
    (..), (..) ],        # additional interfaces
  1,                     # number of VCPUs
  "4,5,6,7",             # VCPU Mask
  "512M"   )             # Guest RAM
# Node-1 (H2)
newNode(1, "H2", NodeType.LINUX_HOST, .....)
# Node-2 (Switch)
newNode(2, "Switch", NodeType.LINUX_OVS, .....)
# Link: H1<-->Switch
newLink(
  (0, 0),                # Node-0, 1st interface
  (2, 0),                # Node-2, 1st interface
  1000,                  # Link speed in Mbps
  0.2   )              # Latency (NetEm params)
# Link: H2<-->Switch
newLink( (1, 0), (2, 1), 1000, 0.2) )
```

Listing 1: A simple network topology: 2x hosts 1x switch

```
# Prevent arp broadcasts
setArp(0,"eth1","10.0.1.3","fe:ff:ff:00:01:03"))
setArp(1,"eth1","10.0.1.2","fe:ff:ff:00:01:02"))
# Configure the switch
pushCmd(2,["ovs-vsctl set-fail-mode br0 secure"])
pushCmd(2,["ovs-vsctl add-port br0 eth1"])
pushCmd(2,["ovs-vsctl add-port br0 eth2"])
pushOFRule(2,
  "br0", "add-flow","in_port=1,action=output:2")
pushOFRule(2,
  "br0", "add-flow","in_port=2,action=output:1")
# Run netperf for 10 seconds
pushCmd(0,
  ["netperf -H 10.0.1.3 -t TCP_STREAM -l10 -D1"])
```

Listing 2: A sample execution scenario description

Step 4 - *Scenario execution*

In the final step, SELENA executes the experimental functionality. This is defined through a separate user-composed Python script, describing a sequence of time-controlled commands to run on each guest. When the scenario is executed, each command is transmitted to the guests through the coordination service (via *Xenstore*).

Listing 2 presents a simple scenario, applied on the network topology in Listing 1. The scenario installs a static datapath between the two ports of the switch and creates static ARP entries to avoid unnecessary broadcast traffic. Then it instructs *H1* to initiate a netperf session and fill the pipe of the virtual path to *H2*. Experiments in SELENA run in dilated virtual time and therefore, scenario commands are also executed in virtual time. For example, a 10 seconds *netperf* session will last multiple times longer in real-time for TDF values higher than 1.

In order to minimize the time it takes to install, configure and start large experiments, we implement a series of optimizations. Firstly, SELENA provides a set of guest templates containing the minimum required software for network experimentation. Our Linux-based guests contain a minimal 3.13 kernel and a stripped-down set of applications and tools, resulting in low memory guest footprint (20 MB) and fast boot times (2-4 seconds). Secondly, SELENA uses the copy-on-write disk cloning functionality of Xen, to decrease the time it takes to replicate a VM's disk image from our templates. Finally, after the completion of an experiment, guests are cached in a guest pool. During *Step-3*, the deployment script checks initially in the pool to find any matching guest VMs created from the same template and reuses it. Nonetheless, the virtual interfaces of guests and the respective bridges in Dom0 are always clean-installed. Using these techniques SELENA can cold-boot, for example, a 52 node topology in $\approx$ 130 seconds.

3.3 Scaling resources via elastic time

An inherent limitation of real-time emulation is resource scalability for high throughput or node-count experiments. For example, emulating the packet-level behavior of a 40 GbE link in real time generates a high event rate and potentially an unmanageable CPU load. Effectively, fidelity is upper-bounded by the ability of the platform to scale the experiment computation and match the emulated system's performance properties. SELENA overcomes this limitation by virtualizing time. In order to achieve this, we implement a time virtualization mechanism in the hypervisor, which allows an experimenter to slow down time progression on guests and effectively scale network, disk IO and CPU resources. Our approach is similar to the *bullet time* video effect: augmented per-frame time enables the viewer to perceive more details from a scene.

With respect to Figure 1, emulated nodes (H_1, H_2, SW) run inside a sandbox using a common time-dilation factor (*TDF*), while Dom0 and the hypervisor operate in real-time. SELENA adjusts all guest time sources by the TDF parameter, *controling* effectively the perception of time-progression both at the kernel and the user-space of the guest VM. Each guest continues to receive a fair fraction of available hardware resources, unaffected by time virtualization. For an experiment with TDF $= 2$, a virtual time of 1 second lasts 2 seconds of real time and the perceived network, disk IO rates and CPU appear doubled within the virtual time domain.

Having created the necessary "computational headroom" to emulate faster IO and processing, the users may chose to further adjust the speed of individual resources for guests. The amount of CPU time that a guest receives can be adjusted via the *weight* and *cap* parameters of Xen's *credit2* scheduler [2] (in non work-conserving mode). Dilating time by a large factor, however, can occasionally introduce transient scheduling inaccuracies. This effect can be minimized by tuning the Xen scheduler parameters, using a scaled *time-slice* and an increased *ratelimit* value (controls the minimum non-preemptive VM scheduling duration). In addition, disk IO rates can be limited from within guests via the *cgroups* mechanism and its *blkio controller*.

Implementation details

SELENA supports time dilation control without requiring modifications in guest OS. Our approach takes advantage of the Linux PVOPS drivers, which provides out of the box in-kernel support for Xen para-virtualized drivers. The resulting modifications required to implement time virtualization were limited only in the Xen hypervisor source code ($\approx$ 400 LoC). We have successfully patched a recent version of the Xen hypervisor (v4.3). In order to avoid xen-api toolstack modifications, we expose TDF control through the sysfs filesystem, using a modified domain creation hypercall ($XEN\_DOMCTL\_createdomain$).

In order to present the functionality of time virtualization in SELENA, we initially describe handling of time in Xen. During boot,

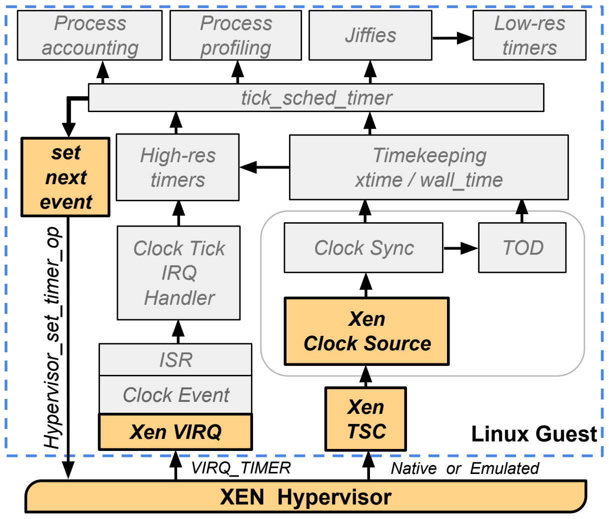


Figure 2: Architecture of the various timekeeping facilities of a Linux PVOPS kernel

the Linux kernel selects the appropriate drivers and clock-sources. Fortunately, the generic time-of-day (GTOD) framework handles transparently a variety of clock sources[3]. Figure 2 shows the Xen-specific components of the timekeeping subsystem. At the bottom reside the low-level bits of the Xen clocksource, a place which typically plays the role of the "bridge" to real hardware. An OS typically requires two basic clock services, *time-keeping* and *time event scheduling*, and therefore, our solution needs to support both.

In terms of time-keeping, Xen exposes to guests two timestamp values, the *system-time* (time since guest boot or resume) and a *wall-clock time* (time since epoch when system-time was zero), through a shared memory structure (`shared_info_page`). *System-time* is updated by Xen every time the guest is being scheduled. In between updates, the guest keeps accounting of wall-clock and system time by extrapolating the current values based on the value of TSC register (an x86 register counting CPU clock cycles). TSC values in Xen are obtained through the `rtdsc` instruction. They are either *native* and read directly from the CPU register or *emulated* by Xen (intercepted through a trap). In order to effectively virtualize time-keeping, the hypervisor multiplies with the TDF parameter all the wall-clock, system and emulated TSC time values provided to the guest. Additionally, because native TSC values are unmodifiable, our approach multiples with the TDF value the TSC scaling factor (*tsc_to_system_mul*), a constant used by the guests to convert TSC cycles to system time.

In terms of time event scheduling, the guest PVOPS driver intercepts and translates such requests to equivalent Xen hypercalls ($HYPERVISOR\_set\_timer\_op$). When the timer expires, the hypervisor delivers a software timer interrupt ($VIRQ\_TIMER$) back to the guest and triggers the associated timekeeping interrupt handlers. Time event virtualization in SELENA is achieved in the hypervisor by intercepting timer-setup hypercalls, scaling their timeout values with the TDF parameter. Our approach also covers the case of periodic timer events used by older Linux version. Recent Linux kernels also employ loop-based delays for timeout values lower than $\sim 10 usec$. These types of events are not directly managed by the hypervisor, but our approach is sufficient to ensure accurate event execution.

[3]FreeBSD also follows a similar approach

3.4 Network Emulation

SELENA emulates network links by creating pairs of guest network interface devices bridged in Dom0. The Xen architecture uses two virtual interfaces (VIFs), the *netfront driver*, exposed to the guest, and the *netback driver*, exposed to Dom0. Netfront and netback interconnect using shared memory rings. This separation of network devices into multiple VIFs introduces a challenge for SELENA to provide good guarantees for network resources, like latency and throughput. Effectively, a link in SELENA must manage at least four independent network queues (see Figure 1), running at different rates, and ensure lossless resource allocation. Ultimately, networking performance effects (e.g. queueing) should occur in the boundaries of the emulated hosts, and not in Dom0.

SELENA's time virtualization provides a useful mechanism to experiment with low latency network environments. High TDF values provide sufficient head-room to minimize packet processing delays and test high-bandwidth setups. For instance, for TDF = 1, the average RTT over an idle guest-to-guest link is approximately 300 *usec*, a value which drops to 30-50 usec for TDF = 10. SELENA uses the netem qdisc [14], a traffic-control primitive in the Linux kernel supporting constant or stochastic latency in network links (*DummyNet* can be used instead in FreeBSD). SELENA configures such qdiscs on the egress queues of the guests, in order to maintain a common time reference between experimental nodes. In recent Linux kernels with high resolution timer support, netem choice provides SELENA with high precision latency control, in the order of sub-milisecond.

In terms of throughput, the process to ensure fidelity is more complex, as throughput can be either CPU or IO bound. SELENA follows a two step empirical method, presented in greater detail in Section 4.4. Firstly, the user tries different TDF values to discover the minimum value which can effectively support the required throughput. Secondly, SELENA applies rate limiting at individual VIFs to match the link speed specifications, defined by the experiment description. Rate limiting uses the QoS primitive of the Xen network driver (*qos_algorithm_type*:*ratelimit*). To ensure accurate network resource scaling, the rate limit of a link is divided by the TDF value of the experiment. For example, a 1 Gbps link translates into a rate limit of 100 Mbps for TDF = 10.

The Xen network driver provides additional control on the trade-off between fidelity and resource scalability. Currently the driver provides multiple network offloading functions which can improve link throughput (e.g. TSO, UFO, TX/RX checksum offload) both on the guests and Dom0 VIFs. When these settings are enabled, we observe that guests can use very large packet sizes (64KB packets), thus improving DomU-to-DomU throughput. Nonetheless, such large packet MTU reduce the packet-level and queue multiplexing fidelity. In the experiments of section 4 we have disabled all these optimization features. Finally, in order for SELENA to support large port-count hosts/switches, we modified the Xen hypervisor source code to increase the size of the page grant table, enabling support up to 48-ports. While raw-throughput is sufficient in this first version of SELENA, opportunities for improvement by bypassing Dom0-constraints are made clear by related work such as VALE [28].

3.5 Data plane emulation

Control planes with programmatic interfaces have redefined the role of network functionalities. The SDN paradigm and its predominant realization, the OpenFlow protocol, define a set of flow-level abstraction primitives, exposed to external entities via a unified programming interface. This has made possible the instantiation of previously complex or impractical network-control ideas.

The growing interest in the SDN paradigm has been a core motivation in recent experimentation platform efforts like Mininet [13]. SELENA supports any software switch implementation which can run in a guest domain. This includes kernel-level bridges, Open vSwitch and other popular software switches (e.g. Click [20]). Nonetheless, recent measurement studies of switch devices [12,30] have highlighted a significant variability on the performance of the control plane, especially in available hardware OpenFlow switch implementations. This performance variability can be explained by a number of design choices, like the scalability of the communication channel between the switch silicon, the switch co-processor and the firmware architecture. However, existing solutions are not easy to reflect these performance properties and can only replicate simplified network device models, considering only per-packet processing latency and buffer sizes.

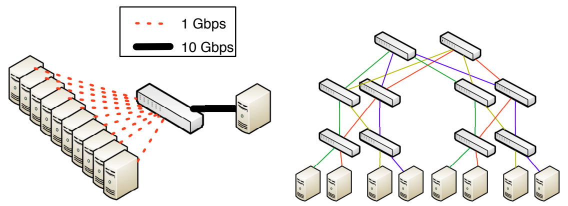
SELENA provides a tunable switch model that takes account of the tight coupling between the flow and management timescales of the network [24]. This allows users to study how traditional and new control plane performance characteristics affect the overall network performance. More specifically, we include a customizable switch implementation which supports OpenFlow [29] and builds on the Mirage framework [23]. This switch emulation model provides a simple and extensible packet processing pipeline, to which we add a rate adjustment mechanism for the different control plane functionalities. Our model exposes currently performance emulation primitives for the flow table management mechanism, the packet interception and injection mechanism and the counters extraction functionality of the switch. Section 4.5 describes a methodology to calibrate a switch model in a SELENA experiment in order to match the performance characteristics of a production switch.

4. EVALUATION

In this section we present an in-depth evaluation of SELENA's performance and scalability. Initially, we introduce the reader to the measurement apparatus of our evaluation (§ 4.1). We compare the experimental fidelity provided by SELENA against a real system and two established experimentation platforms, Mininet and ns-3. Our evaluation considers fidelity both in terms of high-throughput links (§ 4.2), as well as larger networks (§ 4.3). Finally, we present a limit analysis of our time virtualization technique (§ 4.4) and evaluate the performance realism of real applications in SELENA (§ 4.5).

4.1 Experimental Setup

In order to evaluate the performance of SELENA, we use two popular network topologies: a *star topology* (Figure 3(a)) and a



(a) Star topology, 10 clients pushing traffic to a data sink.

(b) Fat-tree topology, 4-port switches, 2 pods and 8 hosts.

Figure 3: Experimental topologies.

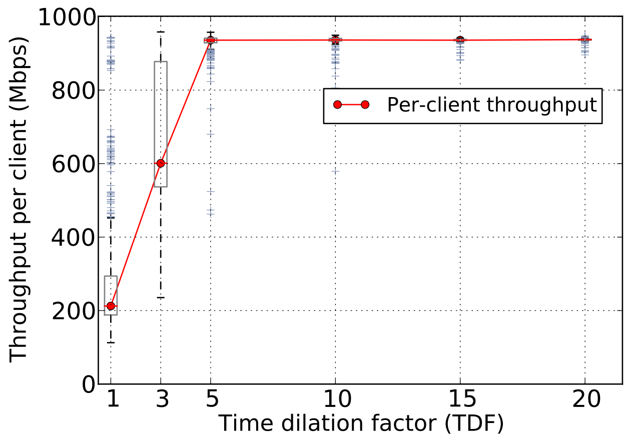


Figure 4: Per-host throughput samples for various TDF values

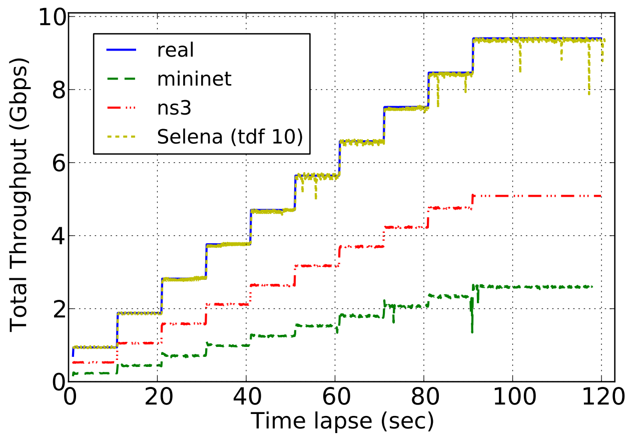


Figure 5: Aggregate achieved throughput between real, SELENA, Mininet and ns-3 (star topology).

special case of a CLOS, a *fat-tree topology* with 4-port switches and 2 pods (Figure 3(b))[4]. We employ these two distinct topologies as they provide a simple mechanism to test both throughput and scalability of the system for high-density setups. We replicate the experiments of each topology four times: in SELENA, in a real setup, in Mininet and in ns-3.

In the real setup, the star topology consists of a quad-core server (Intel Xeon E5-2643, 128GB RAM), 10 data generating hosts (Intel Core 2 Q6600, 4GB) and a network switch (Pica8 P-3290). The server is equipped with a 10GbE card (Solarflare SFC9020) and acts as the data-sink. The clients are equiped with 1 GbE cards (Intel 82571EB) and the switch offers both 1 and 10 GbE interfaces. We monitor the link to the data sink using a DAG card and an optical splitter. For the fat-tree topology we run the end-host logic on the clients of the previous setup, and use 10 NetFPGA-1G equipped hosts with OpenFlow functionality [25], to replicate the switching fabric. For ns-3 we use the `CSMA` link model and build a simple switching host type, to replicate switch behaviour. For the equivalent SELENA experimental setups we use Linux VMs with stock 3.13 kernel and Open vSwitch (v1.11) for switch emulation. SELENA, Mininet and ns-3 are executed on the same quad core server (Intel Xeon E5-2643, 128GB RAM) for fair comparison.

4.2 Data-Plane fidelity

In this section we use the star topology to evaluate the capability of the tested experimentation platforms to support 10 Gbps traffic. For this scenario we use steady-state long-running TCP flows, generated by the netperf application in the real, SELENA and Mininet experiments and a TCP `BulkSendTraffic` application in ns-3. Starting at t=0, a new full-rate TCP-flow is initiated every 10 seconds from a previously idle host, resulting in 10 Gbps throughput at time t=90 sec.

Firstly, we analyse the impact of TDF on the per-client throughput. Figure 4 presents boxplots of the network throughput measurements from all clients, using a monitoring window of 100 msec. We notice that TDF = 5 provides a median throughput of 940Mbps, the maximum achievable TCP rate for a host in this setup. However, this TDF value exhibits a non-negligible variance with a few outliers (grey-colored marks), occurring due to Xen scheduling effects and the inability of the CPU to handle the offered load. This effect is more evident for lower values of TDF (e.g. TDF = 1 achieves a median throughput of 210 Mbps). We conclude that TDF = 10 provides sufficient accuracy and throughput stability for the complete duration of the experiment.

Figure 5 presents the achievable throughput on the 10 Gbps link for the duration of the experiment, using 100 msec observation window. We used TDF = 10 for SELENA which, based on the previous findings, is sufficient to emulate accurately the throughput of the real system. Mininet is able to achieve a maximum of 2.3 Gbps throughput, with non-negligible variance, primarily due to poor scalability of CPU resources. Despite the extensive optimization of Mininet, the platform cannot mitigate CPU and memory bottlenecks in high throughput emulations. ns-3 exhibits higher performance than Mininet achieving a maximum throughput of 4-5 Gbps, but still significantly lower than the real. After analyzing the source code of the ns-3 platform, we concluded that the low aggregate performance is due to poor scalability of the `CSMA` link model at high rate, which doesn't support full-duplex links. Furthermore, ns-3 provides an MPI-based distributed event engine. Nonetheless, this event engine supports only Point-to-Point layer-3 links and its distribution model is restricted by the requirement for clock synchronization.

4.3 Flow-level fidelity

This section evaluates the flow-level fidelity of SELENA using the fat-tree topology. For this experiment we use short-lived TCP flow completion times to assess the accuracy of our platform, a metric which is widely used to characterize both the performance and fairness of TCP in the context of datacenter networks [7]. On each host we run both a data-generation and data-sink service and traffic is flowing in both directions between four pre-defined client pairs (8 Gbps aggregate bandwidth). The data-sink service of each-client receives data from 10 concurrent TCP flows (all form the same source). When a flow transfers 10 MB worth of data it is terminated and a new transfer request is initiated immediately, thus maintaing constantly 10 concurrent flows per host. We implemented this traffic generation model in C using the POSIX API and the *async epoll*-based `libev` API. The generator model is used in SELENA, real and Mininet experiments, while for ns-3 we build a similar application using the TCP abstraction.

Figure 6 presents the empirical cumulative distribution (CDF) of the flow completion times for a wide range of TDF values using SELENA, including also the results collected from the real, Mininet

[4]we employ half of the nodes in a k=4 fat-tree topology, due to luck of physical resources to replicate the real experiment

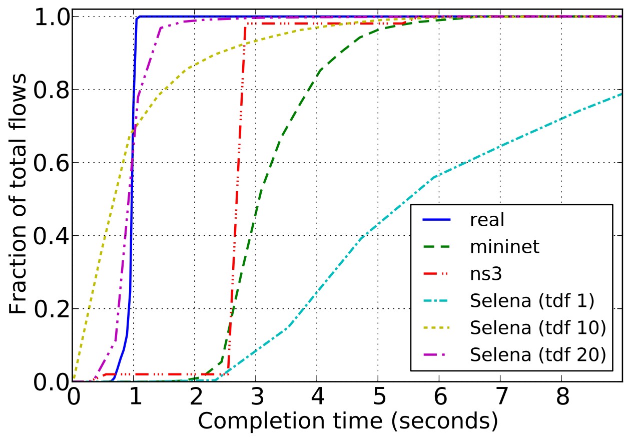


Figure 6: Comparison of flow completion time CDF between SELENA, real, Mininet and ns-3 (fat-tree topology).

and ns-3 platforms. Additionally, Table 2 presents the number of completed flows for the duration of the experiment and the mean and variance of flow completion times for each experiment. For SELENA, we notice that TDF = 1 is highly unrealistic, introducing 6 times higher delay than the real system and significant variance. Nonetheless, as we increase TDF values, the distribution of completion times starts to approximate the behavior of the real system. A TDF = 5 reduces 4.8 times the mean latency, while higher TDF values improve further the experimental fidelity. For TDF = 20 the mean completion time is only 10 msec different from the real system, while the variance is equally reduced.

Mininet increases the mean flow completion times by a factor of 3 and exhibits significant variance, also visible by the longer tail of the cummulative distribution. Note that the majority of the flows in Mininet exhibit completion times between 2.5 and 4.5 seconds. There are also many outliers with a maximum delay of 7 seconds. Mininet clearly cannot support the required aggregate throughput of 8 Gbps using the fat-tree topology and therefore with a lower available bisection capacity flows require more time to complete. ns-3 on the other hand, has an extremely low variance with a median completion time of 2.5 seconds.

Table 3 presents the time scalability properties of the tested platforms, by comparing the wall-clock execution duration of Mininet,

Platform	Flows	Mean	std
Real	10071	0.956	0.074
Mininet	2957	3.271	0.787
SELENA TDF = 1	1512	6.469	3.236
SELENA TDF = 5	7161	1.344	1.144
SELENA TDF = 10	9285	1.036	1.035
SELENA TDF = 20	9935	0.966	0.16

Table 2: Comparison of flow-completion time statistics.

Platform	Star Topology	Fat-tree Topology
Mininet	120s	120s
ns-3	175m 24s	172m 51s
SELENA TDF = 1	120s	120s
SELENA TDF = 20	40m	40m

Table 3: Comparison of wall-clock execution times.

SELENA and ns-3 for both experiments. Mininet runs in real-time and experiment duration is not affected by the complexity of the scenario, it simply sacrifices accuracy. In contrast, ns-3 was considerably slower than SELENA ($\approx$ 4.4x times), despite its poor scaling. This slow-down in execution time stems primarily from its single threaded design and the requirement for total ordering of events. Effectively, the execution speed of an ns-3 experiment is primarily affected by the rate of network events.

4.4 Time-dilation limit analysis

The use of Xen by SELENA, provides high scalability, supporting hundreds of guests [22]. Nonetheless, resource virtualization in Xen uses complex mechanisms to schedule guest vCPU, manage memory, process event channel interrupts, control access to physical hardware (e.g. disk I/O), all of which can significantly affect the accuracy of an experiment. Depending on the emulated network topology and the employed workloads, the impact of system-level effects can vary.

In the star topology (Figure 3(a)), although Dom0 is allocated with 4-cores to relay packets between guests, it can use a maximum of 2.5 cores. To explain this behavior, we need to look into the implementation details of Xen's control domain which maps one kernel-thread per Dom0 core. The 10 Gbps virtual link between the switch and the server has the highest processing demands but its workload cannot be parallelized as it is served from a single netback kernel thread (for one direction) and therefore can utilize only one core. The traffic from the client-nodes to the switch (via Dom0) consumes roughly the same aggregate CPU resources with the 10 Gbps link. Some extra processing takes place also in the network stack and software bridges of the control domain.

In order to present the scalability limits of our time virtualization mechanism, we revisit the star topology experiment (Figure 3(a)) and study the limits of aggregate throughput using TDF = 20. We generate two variations of the experiment: i) 4 clients with variable link speeds and ii) an increasing number of clients each connected through a 1 Gbps link. Figure 7 uses boxplots with outliers (red crosses) to present the measured client throughput (1 second monitoring window) as we increase the capacity of the client links. This figure also illustrates the aggregate CPU utilization for the Dom0 and the nodes of the experiment. We observe that SELENA copes well with client link speeds up to 10Gbps (aggregate sink-server link speed of 40Gbps), evident by the low number of outliers. For client link speeds higher than 10 Gbps, the fidelity degrades as hardware resources do not suffice to serve the offered load. As discussed, the bottleneck of this topology is the sink-server link.

Figure 8, presents a throughput analysis, similar to the previous example, but in this scenario we vary the number of client hosts, keeping link capacity limited to 1 Gbps. For this experiment, SELENA can faithfully emulate network of up to 40 clients with low variance, using TDF = 20. Beyond this number of nodes, per-client median throughput is reduced and the variance increases. Similar to before, the aggregate throughput of the 40 clients is 40Gbps, but we observe more outliers; 2.5% of client throughput observations drop below 750Mbps. This effect can be attributed to Xen's scheduling effects which cause more buffering during the time a guest is not scheduled and therefore more packet losses occur.

In order to provide further evidence on the scalability of SELENA, we evaluate the star topology in terms of throughput and node count for a variety of TDF values. We present the results in Figures 9 and 10, respectively. Each data point represents the maximum throughput or node count which can be reliably supported by a specific TDF value. We consider an experimental result as reli-

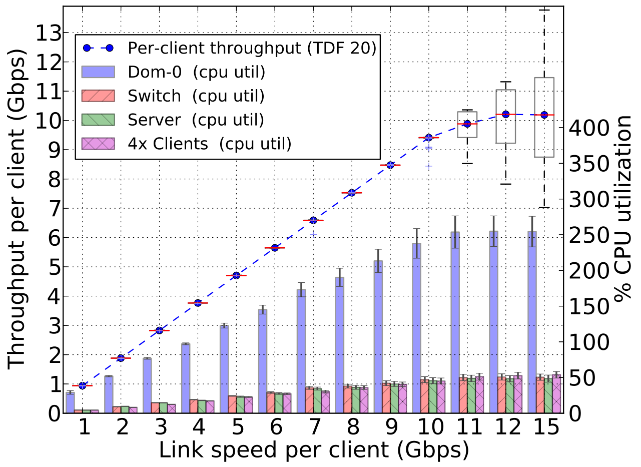


Figure 7: Link speed scalability analysis: per-client throughput and CPU utilisation (star topology, 4x clients)

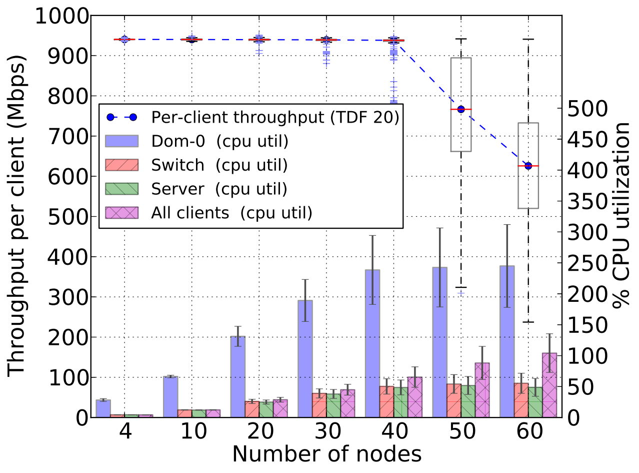


Figure 8: Node-count scalability analysis: per-client throughput and CPU utilisation (star topology, 1 Gbps links)

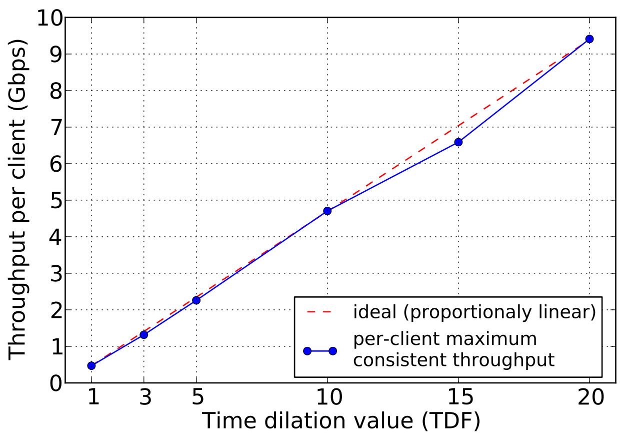


Figure 9: Overall link speed scalability vs TDF

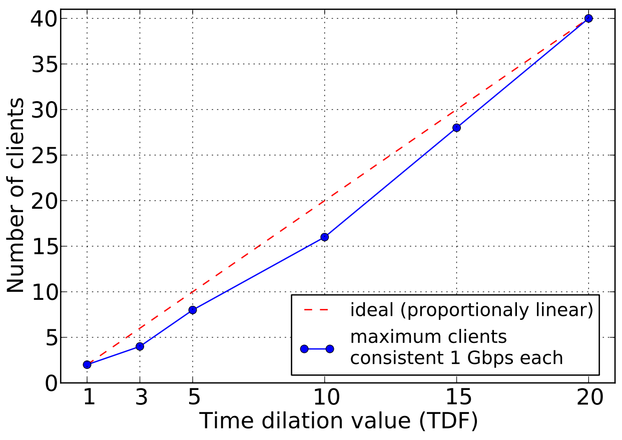


Figure 10: Overall network size scalability vs TDF

able, when the per-host throughput observation achieves the target value for the majority of measurement samples (97.5% of all samples). In both cases we observe that the experiment can achieve close to linear scalability. We should clarify that while we expect SELENA to exhibit a near-linear scalability trend for a wide range of experiments, the user is strongly encouraged to verify this assumption for different virtual topologies individually.

4.5 SELENA applications

In this section we evaluate the ability of SELENA to emulate data plane network behaviors and run real unmodified applications, using a web server deployment.

Control-Plane fidelity

In order to evaluate the data plane fidelity of our switch model, we assess its precision in replicating the performance behavior of a real OpenFlow switch. We calibrate our switch emulation model by measuring the data and control plane performance profile of a production switch (Pica8 P-3290 switch[5]) using the OFLOPS OpenFlow switch evaluation framework. OFLOPS runs on a Quad Core Intel PC (Q6600) equipped with a NetFPGA-1G card directly connected to 4 switch ports and uses the high-precision NetFPGA-1G traffic generation backend.

[5]http://www.pica8.com/

Our characterization effort focuses on the behavior of a reactive control scheme. In this approach, the controller exercises per-flow forwarding decisions, thus achieving fine level of control. In detail, each newly arriving flow generates a packet exception which is propagated to the controller using a `Pkt_in` message. The controller then is responsible to install an appropriate entry in the flow table, which defines a forwarding policy for all matching packets. In order to measure each elementary interaction in this control architecture, we use the OFLOPS measurement modules to evaluate: (1) the delay to install a new flow in the flow table, and (2) the delay and loss rate of `Pkt_in` messages.

Based on the OFLOPS results we observed a median flow insertion delay of 6msec, a `Pkt_in` processing delay of 2msec and a maximum rate of 50 `Pkt_in` messages per second. Using these settings, we configure our switch model and compare its performance with the Pica8 switch and Mininet. We setup a simple topology with two hosts, a data-sink and data generation host replicating the application behavior described in Section 4.3, interconnected through an OpenFlow switch. The data-sink host generates flow requests to the data generator host following an exponential distribution with $\lambda = 0.02$ and a constant request size of 1 MB resulting

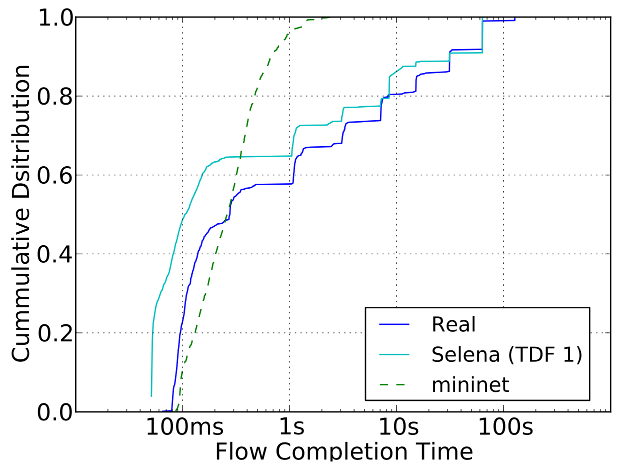


Figure 11: Comparing flow completion times between a real system, Mininet and SELENA

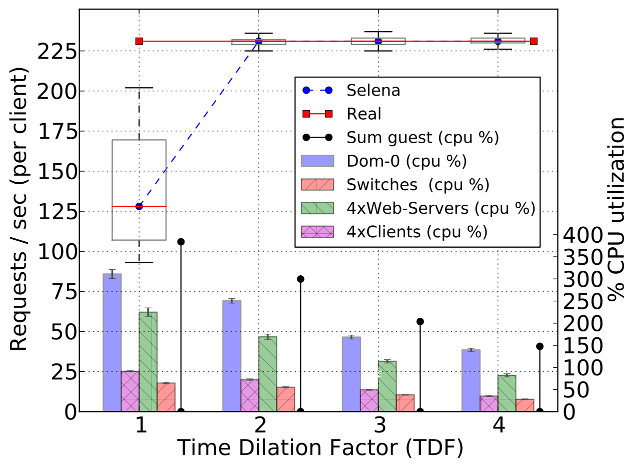
to an average 400 Mbps traffic rate. We run our experiment for 120 seconds, ensuring that the resulting number of flows is below the flow table size limits. For all experiments we used the learning switch application of NOX [9] to control the switch.

In Figure 11 we present the cumulative distribution of flow completion times for the real, SELENA and Mininet experiments. The performance observed with the real switch highlights the impact of the switch control plane design on network performance. In the real experiment, 20% of flows have a completion time higher than 10sec. The predominant cause of this effect is the rate-limiting behavior of the switch in `pkt_in` messages transmissions. This introduces TCP SYN and SYNACK packet drop during bursty periods. As a result, the completion time distribution exhibits a stepping behavior, aligned with the back-off delay mechanism of TCP in the SYN_SENT SYN_RCVD states [6]. The adoption of Open vSwitch by Mininet, allows optimized data-plane performance but it cannot express such device-specific behaviors and thus over-estimates the overall performance of this experiment. Our choice to introduce a customizable model in SELENA enables higher fidelity in OpenFlow experimentation. Our model is able to capture the macroscopic behavior of the switch. Nonetheless, our switch model exhibits a minor over-estimation in completion times. We attribute this behavior to the high load that the reactive control scheme introduces to the communication channel between the co-processor of the switch and the ASIC, which skews the obtained model.

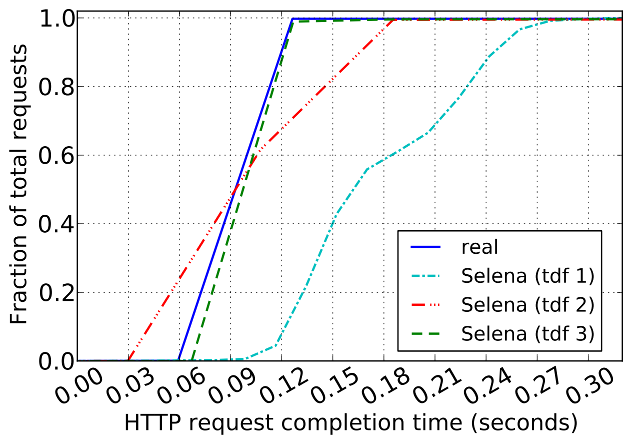
Web server benchmarking

In this section, we evaluate the ability of SELENA to faithfully emulate network examples using a widely-used web application and compare its fidelity against an identical real-world deployment. In this experiment, we reuse the fat-tree topology (Figure 3(b), Section 4.3) and configure the four hosts of the second pod as web-servers, serving a popular dynamic web-application (Wordpress 3.6). Each web service node hosts an Apache web-server (v2.4.6) and a key-value cache (REDIS 2.6.13). We also run a single MySQL (v5.5) database for all Wordpress instances on one of the four hosts. The Wordpress application uses Redis as a temporary cache for generated pages (through the *WP-Redis-Cache* plugin) in order to reduce the processing load on the web servers. On

[6]We use Linux hosts with kernel 3.13, which support TCP Fast Open [6].



(a) Requests throughput: Real vs Selena for various TDF values



(b) Requests completion times:Real vs Selena for various TDF values .

Figure 12: Web application benchmarks: 4x clients, 4x Apache/PHP/Redis (fat-tree)

the first pod, we used the four hosts as clients, each generating 25 parallel web-page requests to a specific web-server, picking a hosted page uniformly at random (approx. size 430-530KB each). The experiment recreation in Selena was identical, precisely replicating application configuration, network topology, link speeds, ram sizes and OS.

Figure 12(a) presents the rate of successfully completed HTTP requests per-client, measured on a per-second basis. The total duration of our experiment is 30 seconds, and for the real deployment, each client manages to achieve a near-uniform rate of ≈ 230 requests/sec (red-colored line). This is close to the theoretical limit of the network speed, implying that our experiment is network-bound. All web-server hosts handled effortlessly the offered load, filling the 1 Gbps pipe (with less than 40% CPU utilization). When running in real-time (TDF = 1), SELENA could not match the observed throughput behavior of the real testbed. In this particular example, Dom0 was not the bottleneck as the CPUs were not fully utilized ($\approx$ 325%) and the packet-forwarding load could be fully parallelized over the four *netback* kernel threads (one per core). The aggregate processing load of guests, however, fully utilized the four assigned to them cores (we run SELENA on an 8-core Intel Xeon server). Once we used higher TDF values, we start observing almost identical throughput results with the real deployment. The higher the time was dilated in guests, the less were utilized the resources of our hosting server, and the more time was required to complete the emulation experiment.

Figure 12(b) illustrates a latency-related aspect of SELENA's fidelity against the real deployment, for various TDF values. In reality, the application performance is bounded by the capacity of the network link, and therefore, the CDF of HTTP-request completion times follows a deterministic trend, adhering to the properties of fair bandwidth-sharing. Using TDF $= 1$ we observe a big deviation in latencies from the real network. This is because computing resources were not enough to cope with the high event rate and Xen also introduced system-level effects (e.g. non-uniform scheduling at a micro-scale), common under high loads. Another interesting observation is that while a value of TDF $= 2$ gives precise enough throughput fidelity, it can not perfectly replicate extremely accurate latency properties, like the value of TDF $= 3$ does.

5. DISCUSSION

In the previous section, we highlighted the achievable fidelity and scalability of the SELENA platform. We now cover a number of important concerns, that we have not covered.

Choosing time dilation factor

SELENA, like all emulation experimentation platforms, provides reproducibility and fidelity on a statistical level. When an experiment is executed multiple times on the same or on heterogeneous platforms, the obtained results should be statistically equivalent. In other words, the distributions of the resulting performance metrics should have high similarity (e.g. using a a test like Kolmogorov - Smirnov). As a result, the design of each experiment must answer two questions: "what is the observed behavior that the experiment replicates" and "what is the required degree of statistical equivalence with a real system". For example, our evaluation considered network latency, throughput and CPU resource availability as behavioral aspects. For the topology in Figure 3(a) (10x clients, 1 GbE links), when emulated using TDF $= 5$ the vast majority of throughput samples are very close (with a few outliers) to the theoretical maximum of 1 GbE for each client. If, however, the user expects 99.9% of throughput measurements to be within a 0.5% error margin ($\approx \pm 4.7 Mbps$), this requires even higher TDF values (15 or more).

From a practical point of view, users must choose a TDF value which provides the desired level of fidelity and also minimize execution time. A useful rule of thumb is to choose a TDF that minimizes the maximum per-CPU utilization. Effectively, experimental fidelity degrades significantly when the experimenter under-provisions computational resources. We currently explore Xen's capabilities, in an effort to automate the process of resource usage monitoring. This will allow SELENA to identify bottlenecks occurring spontaneously by bursty workloads. Handigol *el al.* [13] propose a different approach towards fidelity characterization, monitoring during run-time a set of invariants for network link properties (e.g. inter-packet spacing for non-empty queues). We believe that such an approach is effective for high fidelity biases, capable to detect persistent long-running resource starvation, but it is not clear how it relates to the resulting overall system behavior.

Beyond network emulation

SELENA provides a scalable emulation framework to test network architectures, protocols and modern SDN applications and is best suited for experimental scenarios where system behavior is mandated by network resources. Nonetheless, replicating the behavior of real system can be quite convoluted, especially when its behaviour is significantly affected by minor hardware architecture functional properties of the host. Disk throughput and latency, CPU cache behaviour, per-core lock contention and hardware features, like Intel DDIO [1] are only a few examples which are not easily reproducible in software. Effectively, the design of SELENA is primarily concerned to maximize network fidelity and may not be optimal for experiments that heavily rely on platform-specific characteristics. Furthermore, it is important to highlight that time dilation is not a panacea for high throughput experiments. For example, in order for applications to exploit high capacity links, the experimenter must optimize their configuration (e.g. increase TCP buffers and employ TCP window scaling), while in certain cases the architecture of a service may not be possible to scale for high throughput.

Scaling SELENA

SELENA is based on the Xen Cloud Platform, which can currently scale to a few hundreds of nodes on a server-grade machine [22] [7]. Driven by the observation of the poor performing VIF bridging via Dom0, we explore zero-copy inter-guest network connectivity, similar to [28]. Furthermore, SELENA design can support high density network experiments by distributing the experiment across multiple execution hosts. There are two main challenges for multi-host SELENA execution. Firstly, the experiment must reduce measurement noise incurred by inter-host links and the variable oscillation between the clock sources of the CPUs. Nonetheless, such noise can be minimized using high TDF values. Secondly, the deployment of the experimental nodes between the available execution hosts can be optimized based on the topology and workload of the experiment. Roy *el al.* [31] presented an effective algorithm for the problem, targeting multi-host Mininet execution.

6. CONCLUSIONS

Despite the plethora of available network experimentation tools, the research community is still in the need for solutions which can simplify the testing of new ideas and still provide the necessary level of realism. Common simulation frameworks use simplifying abstraction models, trading accuracy for scalability, and network emulation tools support execution of unmodified code, at the cost of scalability. Motivated by this gap, we have designed and implemented SELENA, a network emulation platform which simplifies experimentation and allows users to trade execution speed for better fidelity as the scale of the experiment increases. Our solution builds on the Xen Cloud Platform and fully-automates the process of deploying and running experiments through an easy to use script-based interface. We have experimentally evaluated the accuracy of SELENA to recreate large and fast network scenarios and compared its fidelity against real deployments and similar experimentation tools. SELENA is open-source and freely available (`http://selena-project.github.io`), hoping that the research community will benefit from our work.

Acknowledgements

This work was jointly supported by the EPSRC INTERNET Project EP/H040536/1 and the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-11-C-0249. The views, opinions, and/or findings

[7]Our evaluation employs medium sized networks, primarily due to our restrictions in host resources, required to compare with a real system.

contained in this article/presentation are those of the author/ presenter and should not be interpreted as representing the official views or policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the Department of Defense.

We would also like to thank Ilias Marinos for his useful comments and his critical views that helped us improve our work.

7. REFERENCES

[1] Intel data direct i/o technology, http://www.intel.co.uk/content/www/uk/en/io/direct-data-i-o.html.

[2] Xen credit scheduler. http://wiki.xen.org/wiki/Credit_Scheduler, 2013.

[3] 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. *ACM SIGOPS Operating Systems Review*, 37, 2003.

[4] S. Bhatia, M. Motiwala, W. Muhlbauer, Y. Mundada, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J. Rexford. Trellis: A platform for building flexible, fast virtual networks on commodity hardware. In *CoNEXT*. ACM, 2008.

[5] M. Carbone and L. Rizzo. Dummynet revisited. *SIGCOMM Comput. Commun. Rev.*, 40(2), Apr. 2010.

[6] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain. TCP Fast Open. Internet-Draft draft-ietf-tcpm-fastopen-05.txt, IETF, Oct. 2013.

[7] N. Dukkipati and N. McKeown. Why flow-completion time is the right metric for congestion control. *SIGCOMM Comput. Commun. Rev.*, 36(1), Jan. 2006.

[8] A. Grau, S. Maier, K. Herrmann, and K. Rothermel. Time jails:a hybrid approach to scalable network emulation. In *PADS*. IEEE, 2008.

[9] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. Nox: Towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, July 2008.

[10] 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. Comp. Syst.*, 29(2), 2011.

[11] D. Gupta, K. Yocum, M. McNett, A. C. Snoeren, A. Vahdat, and G. M. Voelker. To infinity and beyond: time-warped network emulation. In *NSDI*. USENIX, 2006.

[12] M. Gupta, J. Sommers, and P. Barford. Fast, accurate simulation for sdn prototyping. In *HotSDN*. ACM, 2013.

[13] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. Reproducible network experiments using container-based emulation. In *CoNEXT*. ACM, 2012.

[14] S. Hemminger. Netem-emulating real networks in the lab. In *LCA*, Canberra, Australia, 2005.

[15] T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley. Ns-3 project goals. In *WNS2*. ACM, 2006.

[16] T. Issariyakul and E. Hossain. *Introduction to network simulator NS2*. Springer, 2011.

[17] S. Jansen and A. McGregor. Simulation with real world network stacks. In *Simulation Conference, 2005 Winter Proc.* IEEE, 2005.

[18] S. Jansen and A. McGregor. Validation of simulated real world tcp stacks. In *WSC*. IEEE, 2007.

[19] W. Kim, A. Roopakalu, K. Y. Li, and V. S. Pai. Understanding and Characterizing PlanetLab Resource Usage for Federated Network Testbeds. In *IMC*. ACM, 2011.

[20] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, 2000.

[21] A. Köpsel and H. Woesner. Ofelia: Pan-european test facility for openflow experimentation. In *ServiceWave*. Springer-Verlag, 2011.

[22] W. Liu. Improving scalability of xen: The 3000 domains experiment. http://goo.gl/Bt0Gz5, Apr. 2013.

[23] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft. Unikernels: Library operating systems for the cloud. In *ASPLOS*. ACM, 2013.

[24] R. M. Mortier. Multi-timescale internet traffic engineering. *Comm. Mag.*, 40(10), Oct. 2002.

[25] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown. Implementing an OpenFlow Switch on the NetFPGA Platform. In *ANCS*. ACM, 2008.

[26] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. *ACM SIGCOMM Computer Communication Review*, 33(1):59–64, 2003.

[27] Z. Puljiz and M. Mikuc. Imunes based distributed network emulator. In *SoftCOM*, pages 198–203, 2006.

[28] L. Rizzo and G. Lettieri. Vale, a switched ethernet for virtual machines. In *CoNEXT*. ACM, 2012.

[29] C. Rotsos, R. Mortier, A. Madhavapeddy, B. Singh, and A. W. Moore. Cost, performance & flexibility in openflow: Pick three. In *ICC*. IEEE, 2012.

[30] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore. OFLOPS: An Open Framework for OpenFlow Switch Evaluation. In *PAM*, volume 7192. Springer, 2012.

[31] A. R. Roy, M. F. Bari, M. F. Zhani, R. Ahmed, and R. Boutaba. Design and management of dot: A distributed openflow testbed. In *14th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2014.

[32] H. Tazaki, F. Urbani, E. Mancini, M. Lacage, D. Câmara, T. Turletti, W. Dabbous, et al. Direct code execution: revisiting library os architecture for reproducible network experiments. In *CoNEXT*. ACM, 2013.

[33] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. *SIGOPS Oper. Syst. Rev.*, 36(SI), Dec. 2002.

[34] A. Varga and R. Hornig. An Overview of the OMNeT++ Simulation Environment. In *Simutools*, 2008.

[35] E. Weingärtner, F. Schmidt, H. V. Lehn, T. Heer, and K. Wehrle. SliceTime: a platform for scalable and accurate network emulation. In *NSDI*. USENIX, 2011.

[36] N. Willis. Seven problems with linux containers, http://lwn.net/articles/588309/, Feb. 2014.

[37] Y. Zheng and D. M. Nicol. A virtual time system for openvz-based network emulations. In *PADS*. IEEE, 2011.