Reproducible Network Experiments Using Container-Based Emulation

Nikhil Handigol*†, Brandon Heller*†, Vimalkumar Jeyakumar*†, Bob Lantz◇†, Nick McKeown*
[nikhilh,brandonh,jvimal,rlantz,nickm]@[◇cs.]stanford.edu
* Stanford University, Palo Alto, CA USA ◇ Open Networking Laboratory, Palo Alto, CA USA
† These authors contributed equally to this work

ABSTRACT

In an ideal world, all research papers would be *runnable*: simply click to replicate all results, using the same setup as the authors. One approach to enable runnable *network systems* papers is Container-Based Emulation (CBE), where an environment of virtual hosts, switches, and links runs on a modern multicore server, using real application and kernel code with software-emulated network elements. CBE combines many of the best features of software simulators and hardware testbeds, but its performance fidelity is unproven.

In this paper, we put CBE to the test, using our prototype, Mininet-HiFi, to reproduce key results from published network experiments such as DCTCP, Hedera, and router buffer sizing. We report lessons learned from a graduate networking class at Stanford, where 37 students used our platform to replicate 16 published results of their own choosing. Our experiences suggest that CBE makes research results easier to reproduce and build upon.

Categories and Subject Descriptors

C.2.1 [**Computer Systems Organization**]: Computer - Communication Networks—*Network Communications*; D.4.8 [**Operating Systems**]: Performance—*Simulation*

Keywords

Reproducible research, container-based emulation

1. INTRODUCTION

The scientific method dictates that experiments must be reproduced before they are considered valid; in physics and medicine, reproduction is a part of the culture. In computer science, reproduction requires open access to all code, scripts, and data used to produce the results.

CoNEXT'12, December 10–13, 2012, Nice, France.

As Donoho noted,

> *"... a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures."*

Reproducible, runnable papers do not appear to be standard practice in network systems research,[1] so calls from Knuth [1], Claerbout [2], Donoho [3] and Vandewalle [4] for reproducible experiments and results still resonate.

This paper advocates reproducible networking experiments using *Container-Based Emulation*, which runs real code on an emulated network using lightweight, OS-level virtualization techniques combined with careful resource isolation and monitoring. The approach provides the topology flexibility, low cost, and repeatability of simulation with the functional realism of testbeds. The performance fidelity of network emulation is largely unproven, so we focus on exploring the ability of CBE to reproduce experiments by comparing performance results from our prototype, *Mininet-HiFi*, with results published at top-tier networking conferences.

Our specific contributions include the following:

- Implementation of a Container-Based Emulator, Mininet Hi-Fi,[2] which enables reproducible network experiments using resource isolation, provisioning, and monitoring mechanisms (§3).
- Reproduced experiments from published networking research papers, including DCTCP, Hedera, and Sizing Router Buffers (§5).
- Practical lessons learned from unleashing 37 budding researchers in Stanford's *CS244: Advanced Topics in Networking* course upon 13 other published papers (§6).

To demonstrate that network systems research can indeed be made repeatable, each result described in this paper can be repeated by running a single script on an Amazon EC2 [5] instance or on a physical server. Following Claerbout's model, clicking on each figure in the PDF (when viewed electronically) links to instructions to replicate the experiment that generated the figure. We encourage you to put this paper to the test and replicate its results for yourself.

[1](or indeed Computer Science at large)
[2]Available at `https://github.com/mininet`.

	Simulators	Testbeds		Emulators
		Shared	Custom	
Functional Realism		✓	✓	✓
Timing Realism	✓	✓	✓	???
Traffic Realism		✓	✓	✓
Topology Flexibility	✓	(limited)		✓
Easy Replication	✓	✓		✓
Low cost	✓			✓

Table 1: Platform characteristics for reproducible network experiments.

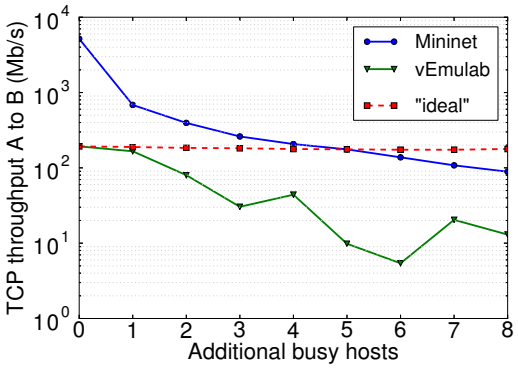


Figure 1: Emulator realism suffers without adequate performance isolation.

2. GOALS

If we are to create *realistic* and *reproducible* networking experiments, then we need a platform with the following characteristics:

Functional realism. The system must have the same functionality as real hardware in a real deployment, and should execute exactly the same code.

Timing realism. The timing behavior of the system must be close to (or indistinguishable from) the behavior of deployed hardware. The system should detect when timing realism is violated.

Traffic realism. The system should be capable of generating and receiving real, interactive network traffic to and from the Internet, or from users or systems on a local network.

In addition to providing realism, the system must make it easy to reproduce results, enabling an entire network experiment workflow – from input data to final results – to be easily created, duplicated, and run by other researchers:

Topology flexibility. It should be easy to create an experiment with any topology.

Easy replication. It should be easy to duplicate an experimental setup and run an experiment.

Low cost. It should be inexpensive to duplicate an experiment, e.g. for students in a course.

Commonly used platforms in networking systems research include *simulators*, *testbeds*, and *emulators*. Table 1 compares how well each platform supports our goals for realism and reproducibility.

Simulators for networking advance virtual time as a result of simulated events [6, 7, 8]. Their experiments are convenient and reproducible, but models for hardware, protocols, and traffic generation may raise fidelity concerns.

Testbeds for networking can be shared among many researchers [9, 10, 11, 12] or specific to one project [13, 14, 15]. Though realistic, they cost money to build and keep running, have practical resource limits (especially before paper deadlines), and may lack the flexibility to support experiments with custom topologies or custom forwarding behavior.

Emulators for networking meet nearly all of our criteria. Like testbeds, emulators run real code (e.g. OS kernel, network applications) with interactive network traffic. Like simulators, they support arbitrary topologies, their virtual "hardware" costs very little, and they can be "shrink-wrapped" with all of their code, configuration and data into disk images to run on commodity virtual or physical machines. *Full-System Emulation*, e.g. DieCast [16] or VMs coupled using Open vSwitch [17], uses one virtual machine per host. *Container-Based Emulation* (CBE), e.g. virtual Emulab (*vEmulab* in this paper) [18], NetKit [19], Trellis [20], CORE [21], Mininet [22] and many others, employs lighter-weight OS-level containers that share a single kernel to achieve better scalability than VM-based systems on a single system [23, 22].

However, emulators, regardless of their type, may not provide adequate performance isolation for experiments. Figure 1 plots the TCP bandwidth for a simple benchmark where two virtual hosts communicate at full speed over a 200Mb/s link. In the background, we vary the load on a number of other (non-communicating) virtual hosts. On Mininet, the TCP flow exceeds the desired performance at first, then degrades gradually as the background load increases. Though vEmulab correctly rate-limits the links, that alone is not sufficient: increasing background load affects the network performance of other virtual hosts, leading to unrealistic results. Ideally, the TCP flow would see a constant throughput of 200Mb/s irrespective of the background load on the other virtual hosts.

For experiments that are limited by network resource constraints (such as bandwidth or latency, as is usually the case for networking research experiments), we conjecture that one can accurately emulate and reproduce experiments on a network of hosts, switches, and links by *carefully allocating and limiting* CPU and link bandwidth, then *monitoring* the experiment to ensure that the emulator is operating "within its limits" and yielding realistic results.

The next section describes the architecture, performance isolation features, and monitoring mechanisms of our Container-Based Emulator, Mininet-HiFi.

3. MININET-HIFI ARCHITECTURE

Mininet-HiFi is a Container-Based Emulator that we have developed to enable repeatable, realistic network experiments. Mininet-HiFi extends the original Mininet architecture [22] by adding mechanisms for performance isolation, resource provisioning, and monitoring for performance fidelity.

3.1 Design Overview

The original Mininet system [22] follows the approach of systems such as Imunes [24] and vEmulab [18] which use lightweight, OS-level virtualization to emulate hosts, switches, and network links. The Linux *container* mechanism (used by Mininet) follows the design of *jails* in BSD and *zones* in Solaris by allowing groups of processes to have independent views of (or *namespaces* for) system resources such as process IDs, user names, file systems and network interfaces, while still running on the same kernel. Containers trade the ability to run multiple OS kernels for lower overhead and better scalability than full-system virtualization.

For each virtual host, Mininet creates a container attached to a *network namespace*. Each network namespace holds a virtual network interface, along with its associated data, including ARP caches and routing tables. Virtual interfaces connect to software switches (e.g Open vSwitch [17]) via virtual Ethernet (`veth`) links. The design resembles a Virtual Machine server where each VM has been replaced by processes in a container attached to a network namespace.

3.2 Performance Isolation

Mininet, as originally implemented, does not provide any assurance of *performance fidelity*, because it does not isolate the resources used by virtual hosts and switches. vEmulab provides a way to limit link bandwidth, but not CPU bandwidth. As we saw earlier, this is insufficient: a realistic emulator requires both CPU and network bandwidth limiting at minimum. In Mininet Hi-Fi, we have implemented these limits using the following OS-level features in Linux:

Control Groups or *cgroups* allow a group of processes (belonging to a container/virtual host) to be treated as a single entity for (hierarchical) scheduling and resource management [25].[3]

CPU Bandwidth Limits enforce a maximum time quota for a `cgroup` within a given period of time [26]. The period is configurable and typically is between 10 and 100 ms. CPU time is fairly shared among all `cgroups` which have not used up their quota (slice) for the given period.

Traffic Control using `tc` configures link properties such as bandwidth, delay, and packet loss.

Figure 2 shows the components of a simple hardware network, and Figure 3 shows its corresponding realization in Mininet-HiFi using the above features.

[3]Resources optionally include CPU, memory, and I/O. CPU caches and Translation Lookaside Buffers (TLBs) cannot currently be managed.

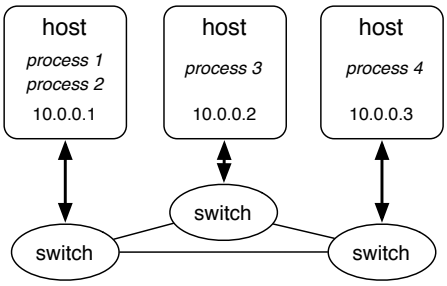


Figure 2: A simple hardware network in a Δ topology.

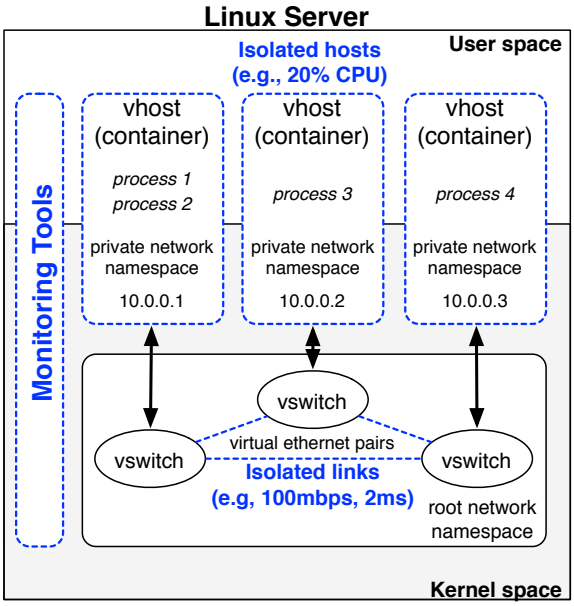


Figure 3: Equivalent Mininet-HiFi network. Dashed lines indicate performance isolation and monitoring features.

Adopting these mechanisms in Mininet-HiFi solves the performance isolation problem shown in Figure 1: the "ideal" line is in fact the measured behavior of Mininet-HiFi.

3.3 Resource provisioning

Each isolation mechanism must be configured appropriately for the system and the experiment. Fortunately, careful provisioning — by splitting the CPU among containers and leaving some margin to handle packet forwarding, based on offline profiling — can yield a result that matches hardware. We report benchmarks comparing Mininet-HiFi with a hardware setup for multiple traffic types (UDP and TCP) and topologies (single link, stars, and trees) in [27].

However, the exact CPU usage for packet forwarding varies with path length, lookup complexity, and link load. It is hard to know in advance whether a particular configuration will provide enough cycles for forwarding. Moreover, it may be desirable to overbook the CPU to support a larger

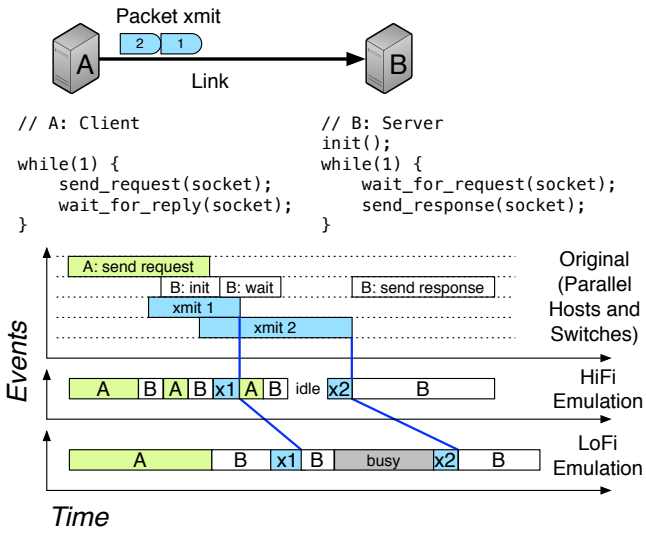


Figure 4: Time multiplexing may cause delayed packet transmissions and host scheduling.

experiment if some links are partially loaded. Mininet-HiFi lets the experimenter allocate link speeds, topologies, and CPU fractions based on their estimated demand, and can also monitor performance fidelity to help verify that an experiment is operating realistically.

3.4 Monitoring Performance Fidelity

Any Container-Based Emulator must contend with infidelity that arises when multiple processes execute serially on the available cores, rather than in parallel on physical hosts and switches. Unlike a simulator running in virtual time, Mininet-HiFi runs in real time and does not pause a host's clock to wait for events. As a result, events such as transmitting a packet or finishing a computation can be delayed due to serialization, contention, and background system load.

Figure 4 compares the timing of a hardware network with two hosts, A and B, sending two packets as part of one request over one link, against emulated versions on a single processor. The bottom part of the figure shows an accurate "HiFi" emulation, along with an inaccurate "LoFi" emulation that has a coarser scheduler granularity and an interfering background process. The slanted lines show that packets in the LoFi version have been delayed and their relative timing has changed, possibly compromising full link throughput and performance fidelity.

To provide assurance that an experiment ran with sufficient fidelity, we want to track when conditions like these occur. Specifically, if the micro-scale scheduling and queueing behavior closely matches hardware, as seen in the HiFi trace, we expect the macro-scale system-level behavior to also match hardware. More precisely, if Mininet-HiFi can (a) start and complete the execution of virtual hosts on time (i.e. within a small delay bound of the corresponding hardware execution times), and (b) dequeue packets on time (i.e. within a small delay bound of the corresponding switch dequeue times), then we can expect high-fidelity results. These are necessary conditions. Since Mininet-HiFi runs inside a single clock domain, it can track the delays accurately.

Link and switch fidelity. We monitor network accuracy by measuring the inter-dequeue times of packets. Since links run at a fixed rate, packets should depart at predictable times whenever the queue is non-empty. For each experiment described in Section 5, we record these samples to determine link and switch fidelity.

Host fidelity. Although CPU bandwidth limiting ensures that no virtual host receives *excessive* CPU time, we also need to determine whether each virtual host is receiving *sufficient* time to execute its workload. We do so by monitoring the fraction of time the CPU is idle. Empirically, we have found the presence of idle time to be a good indicator of fidelity, as it implies that a virtual host is not starved for CPU resources. In contrast, the absence of idle time indicates that the CPU limit has been reached, and we conservatively assume that the emulator has fallen behind, fidelity has been lost, and the experiment should be reconfigured.

Mininet-HiFi uses hooks in the Linux Tracing Toolkit [28] to log these process and packet scheduler events to memory. We next describe the space of experiments suitable for Mininet-HiFi, before testing whether it can actually replicate experiments with high fidelity in §5.

4. EXPERIMENTAL SCOPE

In its current form, Mininet-HiFi targets experiments that (1) are *network-limited* and (2) have aggregate resource requirements that fit within a single modern multi-core server.

Network-limited refers to experiments that are limited by network properties such as bandwidth, latency, and queueing, rather than other system properties such as disk bandwidth or memory latency; in other words, experiments whose results would not change on a larger, faster server. For example, testing how a new version of TCP performs in a specific topology on 100 Mb/s links would be an excellent use of Mininet-HiFi, since the performance is likely to be dependent on link bandwidth and latency. In contrast, testing a modified Hadoop would be a poor fit, since MapReduce frameworks tend to stress memory and disk bandwidth along with the network.

Generally, Mininet-HiFi experiments use less than 100 hosts and links. Experiment size will usually be determined by available CPU cycles, virtual network bandwidth, and memory. For example, on a server with 3 GHz of CPU and 3 GB RAM that can provide 3 Gb/s of internal packet bandwidth, one can create a network of 30 hosts with 100 MHz CPU and 100 MB memory each, connected by 100 Mb/s links. Unsurprisingly, this configuration works poorly for experiments that depend on several 1 Gbps links.

Overall, Mininet-HiFi is a good fit for experiments that benefit from flexible routing and topology configuration and

have modest resource requirements, where a scaled-down version can still demonstrate the main idea, even with imperfect fidelity. Compared with hardware-only testbeds [9, 10, 11, 12], Mininet-HiFi makes it easier to reconfigure the network to have specific characteristics, and doesn't suffer from limited availability before a conference deadline. Also, if the goal is to scale out and run hundreds of experiments at once, for example when conducting a massive online course or tutorial, using Mininet-HiFi on a public cloud such as Amazon EC2 or on the laptops of individual participants solves the problem of limited hardware resources.

If an experiment requires extremely precise network switching behavior, reconfigurable hardware (e.g. NetFPGAs) may be a better fit; if it requires "big iron" at large scale, then a simulator or testbed is a better choice. However, the Container-Based Emulation approach is not fundamentally limited to medium-scale, network-limited experiments. The current limitations could be addressed by (1) expanding to multiple machines, (2) slowing down time [29], and (3) isolating more resources using Linux Containers [30].

The next sections (§5 and §6) show the range of network-limited network experiments that Mininet-HiFi appears to support well. All are sensitive to bandwidth, queues, or latency, and, to generate enough traffic to saturate links, are necessarily sensitive to raw CPU.

5. EXPERIMENTAL EVALUATION

To evaluate and demonstrate Mininet-HiFi, this section details several reproducible experiments based on published networking research. The first goal is to see whether results measured on Mininet-HiFi can qualitatively match the results generated on hardware for a range of network-limited network experiments. The second goal is to see whether the monitoring mechanisms in Mininet-HiFi can indicate the fidelity of an experiment.

Each published result originally used a custom testbed, because there was no shared testbed available with the desired characteristics; either the experiment required custom packet marking and queue monitoring (§5.1), a custom topology with custom forwarding rules (§5.2), or long latencies and control over queue sizes (§5.3). Each Mininet-HiFi result uses an Intel Core i7 server with four 3.2 GHz cores and 12 GB of RAM. If the corresponding results from the testbed and server match, then perhaps the testbed was unnecessary.

For each experiment, we link to a "runnable" version; clicking on each figure in the PDF (when viewed electronically) links to instructions to replicate the experiment.

5.1 DCTCP

Data-Center TCP was proposed in SIGCOMM 2010 as a modification to TCP's congestion control algorithm [14] with the goal of simultaneously achieving high throughput and low latency. DCTCP leverages the Explicit Congestion Notification [31] feature in commodity switches to detect and react not only to the presence of network congestion but also to its *extent*, measured through the sequence of ECN marks stamped by the switch.

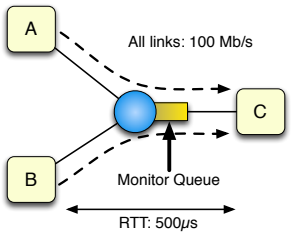


Figure 5: Topology for TCP and DCTCP experiments.

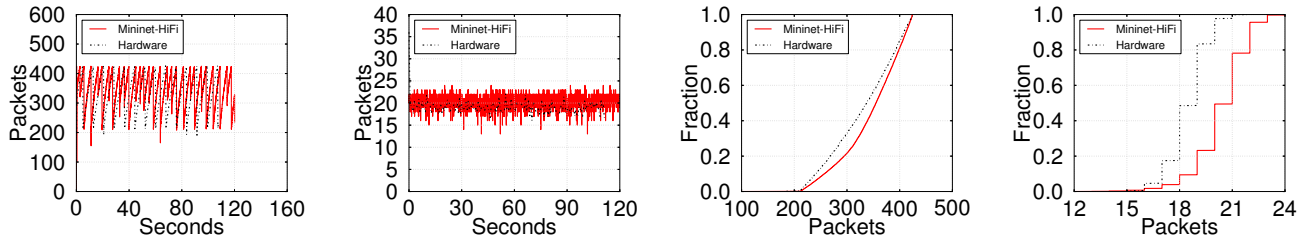
To test the ability of Mininet-HiFi to precisely emulate queues, we attempt to replicate an experiment in the DCTCP paper showing how DCTCP can maintain high throughput with very small buffers. We use the same publicly available Linux DCTCP patch [32]. In both Mininet-HiFi and on real hardware in our lab,[4] we created a simple topology of three hosts A, B and C connected to a single 100 Mb/s switch, as shown in Figure 5. In Mininet-HiFi, we configured ECN through Linux Traffic Control's RED queuing discipline and set a marking threshold of 20 packets.[5] Hosts A and B each start one long-lived TCP flow to host C. We monitor the instantaneous output queue occupancy of the switch interface connected to host C. Figure 6 shows the queue behavior in Mininet-HiFi running DCTCP and from an equivalently configured hardware setup. Both TCP and DCTCP show similar queue occupancies in Mininet-HiFi and hardware, with a bit more variation in Mininet-HiFi. The main takeaway is that this experiment could be emulated using Mininet-HiFi.

Verifying fidelity: DCTCP's dynamics depend on queue occupancy at the switch, so the experiment relies on accurate link emulation. As described earlier, Mininet-HiFi verifies the link emulation accuracy by monitoring the dequeue times on every link. An ideal 100 Mb/s link would take 121.1 μs to transmit a 1514 byte packet, while a 1 Gb/s link would take 12.11 μs. We compute the percentage deviation of inter-dequeue times from the ideal (121.1 μs for 100 Mb/s). That is, if x_i is a sample, the percentage deviation is $100 \times |x_i - 121.1|/121.1$. Figure 7 shows the complementary CDF when we emulate links at 100 Mb/s and 1 Gb/s.

The `htb` link scheduler emulates 100 Mb/s well: the inter-dequeue times are within 10% of an ideal link for 99% of packets observed in a 2 s time window, and within 1% for $\sim$ 90% of packets. The scheduler falls behind for 1 Gb/s links. The inter-dequeue time deviations are far from ideal for over 10% of packets in the same 2 s time window. Though not shown, the average bottleneck link utilization (over a period of 1 s) drops to $\sim$ 80% of the what was observed on hardware, and the CPU shows no idle time. This tells the

[4]We used the same Broadcom switch as the authors.

[5]As per [14], 20 packets exceeds the theoretical minimum buffer size to maintain 100% throughput at 100 Mb/s.



(a) TCP: Instantaneous queue occupancy. (b) DCTCP: Instantaneous queue occupancy. (c) TCP: CDF of instantaneous queue occupancy. (d) DCTCP: CDF of instantaneous queue occupancy.

Figure 6: Reproduced results for DCTCP [14] with Mininet-HiFi and a identically configured hardware setup. Figure 6(b) shows that the queue occupancy with Mininet-HiFi stays within 2 packets of hardware.

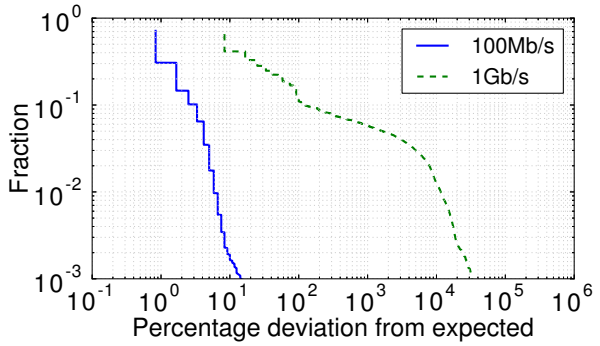


Figure 7: Complementary CDF of inter-dequeue time deviations from ideal; high fidelity at 100 Mb/s, low fidelity at 1 Gb/s.

experimenter that, for this experiment, Mininet-HiFi can emulate the links accurately at 100 Mb/s, but not at 1 Gb/s.

Scaling the experiment: The DCTCP paper showed experimental results for 1 Gb/s and 10 Gb/s bottleneck links, but Mininet-HiFi could only emulate up to a 100 Mb/s link. We expect 1 Gb/s would be attainable with some effort, but 10 Gb/s seems unlikely in the near term. A 10 Gb/s link dequeues packets every 1.2 μs, which stretches the limits of today's hardware timers. In particular, we found that the best timer resolution offered by Linux's High Resolution Timer (`hrtimer`) subsystem was about 1.8 μs, whose limits depend on the frequency of the hardware clock, and its interrupt and programming overheads.

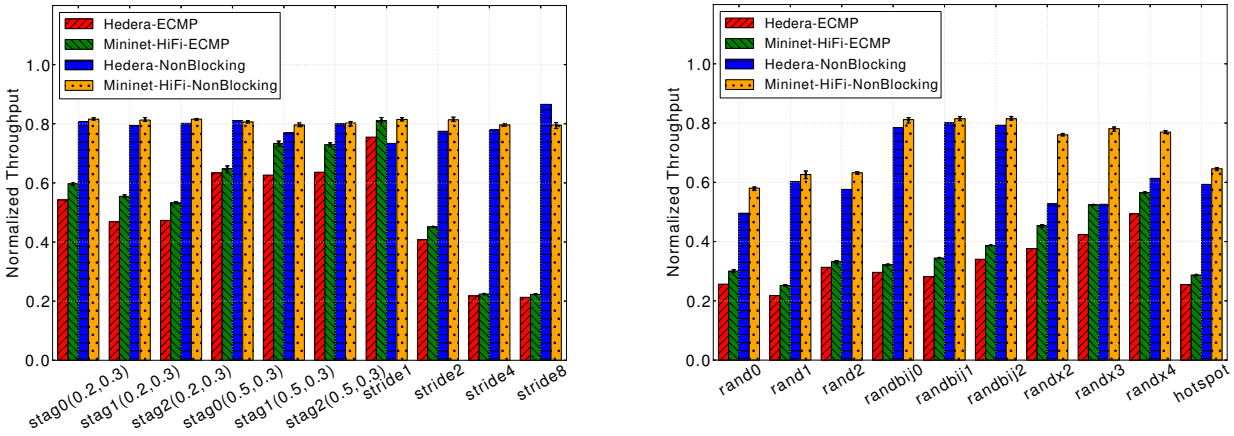
5.2 Hedera

In our second example we use Mininet-HiFi to reproduce results that were originally measured on a real hardware testbed in *Hedera* [13], a dynamic flow scheduler for data center networks, presented at NSDI 2010. With Equal-Cost Multi-Path (ECMP) routing, flows take a randomly picked path through the network based on a hash of the packet header. ECMP hashing prevents packet reordering by ensuring all packets belonging to a flow take the same path [33]. Hedera shows that this simple approach leads to random hash collisions between "elephant flows" – flows that are a large fraction of the link rate – causing the aggregate throughput to plummet. With this result as the motivation, Hedera proposes a solution to intelligently re-route flows to avoid collisions, and thus, exploit all the available bandwidth.

More specifically, as part of the evaluation, the authors compare the throughput achieved by ECMP with that of an ideal "non-blocking" network (the maximum achievable) for 20 different traffic patterns (Figure 9 in the original paper [13]). The authors performed their evaluation on a hardware testbed with a $k = 4$ Fat Tree topology with 1 Gb/s links. The main metric of interest is the aggregate throughput relative to the full bisection bandwidth of the network.

To test the ability of Mininet-HiFi to emulate a complex topology with many links, switches, and hosts, we replicate the ECMP experiment from the paper. We use the same $k = 4$ Fat Tree topology and the same traffic generation program provided by the Hedera authors to generate the same traffic patterns. To route flows, we use RipL-POX [34], a Python-based OpenFlow controller. We set the link bandwidths to 10 Mb/s and allocate 25% of a CPU core on our eight core machine to each of 16 hosts (i.e. a total of 50% load on the CPU). We set the buffer size of each switch to 50 packets per port, our best estimate for the switches used in the hardware testbed.

Figure 8 shows the normalized throughput achieved by the two routing strategies – ECMP and non-blocking – with Mininet-HiFi, alongside results from the Hedera paper for different traffic patterns.The Mininet-HiFi results are averaged over three runs. The traffic patterns in Figure 8(a) are all bijective; they should all achieve maximum throughput for a full bisection bandwidth network. This is indeed the case for the results with the "non-blocking" switch. The throughput is lower for ECMP because hash collisions decrease the overall throughput. We can expect more collisions if a flow traverses more links. All experiments show the same behavior, as seen in the `stride` traffic patterns. With increasing stride values (1, 2, 4 and 8), flows traverse more layers, decreasing throughput.



(a) Benchmark tests from Hedera paper (Part 1).

(b) Benchmark tests from Hedera paper (Part 2).

Figure 8: Effective throughput with ECMP routing on a $k = 4$ Fat Tree vs. an equivalent non-blocking switch. Links are set to 10 Mb/s in Mininet-HiFi and 1 Gb/s in the hardware testbed [13].

Remark: The ECMP results obtained on the Hedera testbed and Mininet-HiFi differed significantly for the `stride-1, 2, 4 and 8` traffic patterns shown in Figure 8(a). At higher stride levels that force all traffic through core switches, the aggregate throughput with ECMP should reduce. However, we found the extent of reduction reported for the Hedera testbed to be exactly consistent with spanning-tree routing results from Mininet-HiFi. We postulate that a misconfigured or low-entropy hash function may have unintentionally caused this style of routing. After several helpful discussions with the authors, we were unable to explain a drop in the ECMP performance reported in [13]. Therefore, we use spanning tree routing for Mininet-HiFi Hedera results for *all* traffic patterns. For consistency with the original figures, we continue to use the "ECMP" label.

The Mininet-HiFi results closely match those from the hardware testbed; in 16 of the 20 traffic patterns they are nearly identical. In the remaining four traffic patterns (`randx2,3,4` and `stride8`) the results in the paper have lower throughput because – as the authors explain – the commercial switch in their testbed is built from two switching chips, so the total buffering depends on the traffic pattern. To validate these results, we would need to know the mapping of hosts to switch ports, which is unavailable.

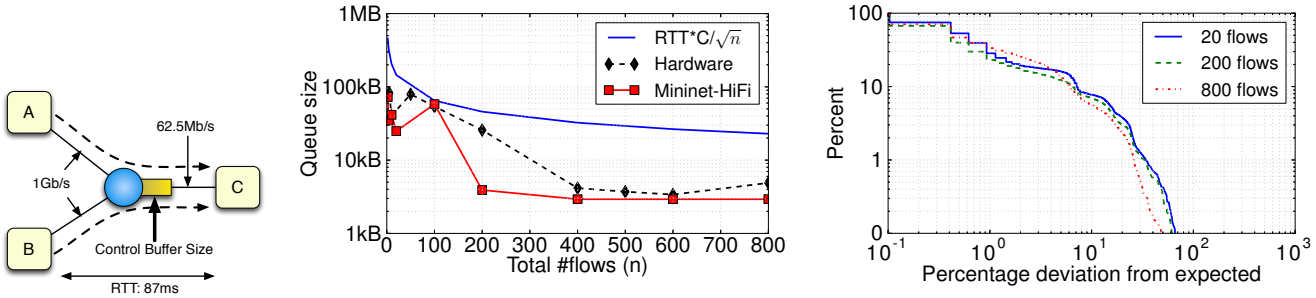
The main takeaway from this experiment is that Mininet-HiFi reproduces the *performance* results for this set of datacenter networking experiments. It appears possible to collect meaningful results in advance of (or possibly without) setting up a hardware testbed. If a testbed is built, the code and test scripts used in Mininet-HiFi can be reused without change.

Verifying fidelity: Unlike DCTCP, the Hedera experiment depends on coarse-grained metrics such as aggregate throughput over a period of time. To ensure that no virtual host starved and that the system had enough capacity to sustain the network demand, we measured idle time during the experiment (as described in §3.4). In all runs, the system had at least 35% idle CPU time every second. This measurement indicates that the OS was able to schedule all virtual hosts and packet transmissions without falling behind an ideal execution schedule on hardware.

Scaling the experiment: In the Hedera testbed, machines were equipped with 1 Gb/s network interfaces. We were unable to use Mininet-HiFi to replicate Hedera's results even with 100 Mb/s network links, as the virtual hosts did not have enough CPU capacity to saturate their network links. While Hedera's results do not qualitatively change when links are scaled down, it is a challenge to reproduce results that depend on the absolute value of link/CPU bandwidth.

5.3 Sizing Router Buffers

In our third example we reproduce results that were measured on a real hardware testbed to determine the number of packet buffers needed by a router. The original research paper on *buffer sizing* was presented at SIGCOMM 2004 [35]. All Internet routers contain buffers to hold packets during times of congestion. The size of the buffers is dictated by the dynamics of TCP's congestion control algorithm: the goal is to make sure that when a link is congested, it is busy 100% of the time, which is equivalent to making sure the buffer never goes empty. Prior to the paper, the common assumption was that each link needs a buffer of size $B = RTT \times C$, where RTT is the average round-trip time of a flow passing across the link and C is the data-rate of the bottleneck link. The authors showed that a link with n flows requires no more than $B = \frac{RTT \times C}{\sqrt{n}}$. The original paper included results from simulation and measurements from a real router, but not for a real network. Later, at SIGCOMM 2008, this result was demon-



(a) Experiment topology.

(b) Results: Buffers needed for 99% utilization, comparing results from the Internet2 testbed, Mininet-HiFi, and the theoretical upper bound.

(c) Verifying Fidelity: A large number of flows increases the inter-dequeue time deviations only by 40% from that of an ideal 100% utilized link, and only for 1% of all packets in a 2s time window.

Figure 9: Buffer sizing experiment.

strated on a hardware testbed running on the Internet2 backbone.[6]

To test the ability of Mininet-HiFi to emulate hundreds of simultaneous, interacting flows, we attempt to replicate this hardware experiment. We contacted the researchers and obtained results measured on their hardware testbed, then compared them with results from Mininet-HiFi; the Mininet-HiFi topology is shown in Figure 9(a). In the hardware experiments, a number of TCP flows go from a server at Stanford University (California) to at a server at Rice University (Houston, Texas) via a NetFPGA [36] IPv4 router in the Internet2 POP in Los Angeles. The link from LA to Houston is constrained to 62.5 Mb/s to create a bottleneck, and the end-to-end RTT was measured to be 87 ms. Once the flows are established, a script runs a binary search to find the buffer size needed for 99% utilization on the bottleneck link. Figure 9(b) shows results from theory, hardware, and Mininet-HiFi. Both Mininet-HiFi and hardware results are averaged over three runs; on Mininet-HiFi, the average CPU utilization did not exceed 5%.

Both results are bounded by the theoretical limit and confirm the new rule of thumb for sizing router buffers. Mininet-HiFi results show similar trends to the hardware results, with some points being nearly identical. If Mininet-HiFi had been available for this experiment, the researchers could have gained additional confidence that the testbed results would match the theory.

Verifying fidelity: Like the DCTCP experiment, the buffer sizing experiment relies on accurate link emulation at the bottleneck. However, the large number of TCP flows increases the total CPU load. We visualize the effect of system load on the distribution of deviation of inter-dequeue times from that of an ideal link. Figure 9(c) plots the CDF of deviations (in percentage) for varying numbers of flows. Even for 800 flows, more than 90% of all packets in a 2 s time interval were dequeued within 10% of the ideal dequeue time (of 193.8 μs for full-sized 1514 byte packets). Even though inter-dequeue times were off by 40% for 1% of all packets, results on Mininet-HiFi qualitatively matched that of hardware.

Scaling the experiment: The experiment described in the original paper used multiple hosts to generate a large number of TCP flows. To our surprise, we found that a single machine was capable of generating the same number (400–800) of flows and emulating the network with high fidelity. While results on Mininet-HiFi qualitatively matched hardware, we found that the exact values depended on the version of the kernel (and TCP stack).

6. PRACTICAL EXPERIENCES

After successfully replicating several experiments with Mininet-HiFi, the next step was to attempt to reproduce as broad a range of networking research results as possible, to learn the limits of the approach as well as how best to reproduce others' results. For this task we enlisted students in CS244, a masters-level course on Advanced Topics in Networking in Spring quarter 2012 at Stanford, and made reproducing research the theme of the final project. In this section, we describe the individual project outcomes along with lessons we learned from the assignment.

6.1 Project Assignment

The class included masters students, undergraduate seniors, and remote professionals, with systems programming experience ranging from a few class projects all the way to years of Linux kernel development. We divided the class of 37 students into 18 teams (17 pairs and one triple).

For their final project, students were given a simple, open-ended request: choose a published networking research paper and try to replicate its primary result using Mininet-HiFi on an Amazon EC2 instance. Teams had four weeks: one week to choose a paper, and three weeks to replicate it. Amazon kindly donated each student $100 of credit for use on

[6]Video of demonstration at http://www.youtube.com/watch?v=ykga6N_x27w.

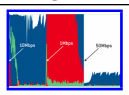
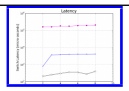
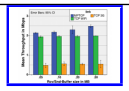
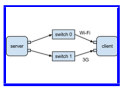
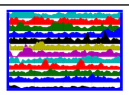
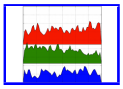
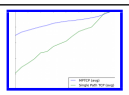
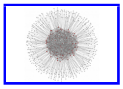
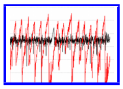
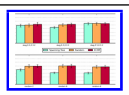
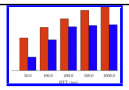
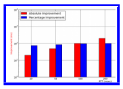
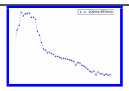
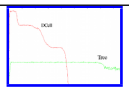
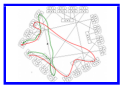
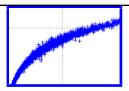
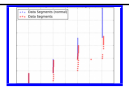
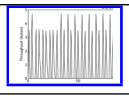
Project	Image	Result to Replicate	Outcome
CoDel [37]		The Controlled Delay algorithm (CoDel) improves on RED and tail-drop queueing by keeping delays low in routers, adapting to bandwidth changes, and yielding full link throughput with configuration tweaking.	replicated + extra results
HULL [38]		By sacrificing a small amount of bandwidth, HULL can reduce average and tail latencies in data center networks.	replicated
MPTCP [39]		Multipath TCP increases performance over multiple wireless interfaces versus TCP and can perform seamless wireless handoffs.	replicated
		Over 3G and WiFi, optimized Multipath TCP with at least 600 KB of receive buffer can fully utilize both links.	replicated, w/differences
Outcast [40]		When TCP flows arrive at two input ports on a tail-drop switch and compete for the same output port, the port with fewer flows will see vastly degraded throughput.	replicated
		The problem described for Outcast [40] occurs in a topology made to show the problem, as well as in a Fat Tree topology, and routing style does not necessarily alleviate the issue.	replicated + extra results
Jellyfish [41]		Jellyfish, a randomly constructed network topology, can achieve good fairness using k-shortest paths routing, comparable to a Fat Tree using ECMP routing, by using MPTCP.	replicated + extra results
		Jellyfish, a randomly constructed network topology, can achieve similar and often superior average throughput to a Fat Tree.	replicated
DCTCP [14]		Data Center TCP obtains full throughput with lower queue size variability than TCP-RED, as long as the ECN marking threshold K is set above a reasonable threshold.	replicated
Hedera [13]		The Hedera data center network flow scheduler improves on ECMP throughput in a $k = 4$ Fat Tree, and as the number of flows per host increase, the performance gain of Hedera decreases.	replicated
Init CWND [42]		Increasing TCP's initial congestion window can significantly improve the completion times of typical TCP flows on the Web.	replicated
		Increasing TCP's initial congestion window tends to improve performance, but under lossy network conditions an overly large initial congestion window can hurt performance.	replicated
Incast [43]		Barrier-synchronized TCP workloads in datacenter Ethernets can cause significant reductions in application throughput.	unable to reproduce
DCell [44]		DCell, a recursively-defined data center network topology, provides higher bandwidth under heavy load than a tree topology.	replicated + extra results
		The routing algorithm used for DCell can adapt to failures.	replicated
FCT [45]		The relationship between Flow Completion Time (FCT) and flow size is not ideal for TCP; small flows take disproportionately long.	replicated
TCP Daytona [46]		A misbehaving TCP receiver can cause the TCP sender to deliver data to it at a much higher rate than its peers.	replicated
RED [47]		Random Early Detection (RED) gateways keep average queue size low while allowing occasional bursts of packets.	unable to reproduce

Table 2: Student projects for CS244 Spring 2012, in reverse chronological order. Each project was reproduced on Mininet-HiFi on an EC2 instance. The image for each project links to a full description, as well as instructions to replicate the full results, on the class blog: http://reproducingnetworkresearch.wordpress.com.

EC2. Each team created a blog post describing the project, focusing on a single question with a single result, with enough figures, data, and explanation to convince a reader that the team had actually reproduced the result – or discovered a limitation of the chosen paper, EC2, or Mininet-HiFi. As an added wrinkle, each team was assigned the task of running another team's project to reproduce their results, given only the blog post.

6.2 Project Outcomes

Table 2 lists the teams' project choices, the key results they tried to replicate, and the project outcomes. If you are viewing this paper electronically, clicking on each experiment image in the table will take you to the blog entry with instructions to reproduce the results. Students chose a wide range of projects, covering transport protocols, data center topologies, and queueing: MPTCP [48, 39], DCTCP [14], Incast [43], Outcast [40], RED [47], Flow Completion Time [45], Hedera [13], HULL [38], Jellyfish [41], DCell [44], CoDel [37], TCP Initial Congestion Window [42], and Misbehaving TCP Receivers [46]. In eight of the eighteen projects, the results were so new that they were only published after the class started in April 2012 (MPTCP Wireless, Jellyfish, HULL, TCP Outcast, and CoDel).

After three weeks, 16 of the 18 teams successfully reproduced at least one result from their chosen paper; only two teams could not reproduce the original result.[7] Four teams added new results, such as understanding the sensitivity of the result to a parameter not in the original paper. By "reproduced a result", we mean that the experiment may run at a slower link speed, but otherwise produces qualitatively equivalent results when compared to the results in the papers from hardware, simulation, or another emulated system. For some papers, the exact parameters were not described in sufficient detail to exactly replicate, so teams tried to match them as closely as possible.

All the project reports with the source code and instructions to replicate the results are available at `reproducingnetworkresearch.wordpress.com`, and we encourage the reader to view them online.

6.3 Lessons Learned

The most important thing we learned from the class is that paper replication with Mininet-HiFi on EC2 is reasonably *easy*; students with limited experience and limited time were able to complete a project in four weeks. Every team successfully validated another team's project, which we credit to the ability to quickly start a virtual machine in EC2 and to share a disk image publicly. All experiments could be repeated by another team in less than a day, and most could be repeated in less than an hour.

[7]The inability to replicate RED could be due to bugs in network emulation, a parameter misconfiguration, or changes to TCP in the last 20 years; for Incast, we suspect configuration errors, code errors, or student inexperience.

The second takeaway was the breadth of replicated projects; Table 2 shows that the scope of research questions for which Mininet HiFi is useful extends from high-level topology designs all the way down to low-level queueing behavior. With all projects publicly available and reproducible on EC2, the hurdle for extending, or even understanding these papers, is lower than before.

When was it easy? Projects went smoothly if they primarily required configuration. One example is TCP Outcast. In an earlier assignment to learn basic TCP behavior, students created a parking-lot topology with a row of switches, each with an attached host, and with all but one host sending to a single receiver. Students could measure the TCP sawtooth and test TCP's ability to share a link fairly. With many senders, the closest sender to the receiver saw lower throughput. In this case, simply configuring a few `iperf` senders and monitoring bandwidths was enough to demonstrate the TCP outcast problem, and every student did this inadvertently.

Projects in data center networking such as Jellyfish, Fat Tree, and DCell also went smoothly, as they could be built atop open-source routing software [49, 34]. Teams found it useful to debug experiments interactively by logging into their virtual hosts and generating traffic. A side benefit of writing control scripts for emulated (rather than simulated) hosts is that when the experiment moves to the real world, with physical switches and servers, the students' scripts can run without change [22].

When was it hard? When kernel patches were unavailable or unstable, teams hit brick walls. XCP and TCP Fast Open kernel patches were not available, requiring teams to choose different papers; another team wrestled with an unstable patch for setting microsecond-level TCP RTO values. In contrast, projects with functioning up-to-date patches (e.g. DCTCP, MPTCP, and CoDel) worked quickly. Kernel code was not strictly necessary – the Misbehaving TCP Receivers team modified a user-space TCP stack – but kernel code leads to higher-speed experiments.

Some teams reported difficulties scaling down link speeds to fit on Mininet-HiFi if the result depended on parameters whose dependence on link speed was not clear. For example, the Incast paper reports results for one hardware queue size and link rate, but it was not clear when to expect the same effect with slower links, or how to set the queue size [43]. In contrast, the DCTCP papers provided guidelines to set the key parameter K (the switch marking threshold) as a function of the link speed.

7. RELATED WORK

Techniques and platforms for network *emulation* have a rich history, and expanded greatly in the early 2000s. Testbeds such as PlanetLab [10] and Emulab [12] make available large numbers of machines and network links for researchers to programmatically instantiate experiments. These platforms use tools such as NIST Net [50], Dummynet [51], and `netem` [52], which each configure net-

work link properties such as delays, drops and reordering. Emulators built on full-system virtualization [53], like DieCast [16] (which superseded ModelNet [54]) and several other projects [55], use virtual machines to realistically emulate end hosts. However, VM size and overhead may limit scalability, and variability introduced by hypervisor scheduling can reduce performance fidelity [56].

To address these issues, DieCast uses *time dilation* [29], a technique where a hypervisor slows down a VM's perceived passage of time to yield effectively faster link rates and better scalability. SliceTime [56] takes an alternate *synchronized virtual time* approach – a hybrid of emulation and simulation that trades off real-time operation to achieve scalability and timing accuracy. SliceTime runs hosts in VMs and synchronizes time between VMs and simulation, combining code fidelity with simulation control and visibility. FPGA-based simulators have demonstrated the ability to replicate data center results, including TCP Incast [43], using simpler processor cores [57].

The technique of *container-based virtualization* [23] has become increasingly popular due to its efficiency and scalability advantages over full-system virtualization. Mininet [22], Trellis [20], IMUNES [24], vEmulab [18], and Crossbow [58] exploit lightweight virtualization features built for their respective OSes. For example, Mininet uses Linux containers [30, 25], vEmulab uses FreeBSD jails, and IMUNES uses OpenSolaris zones.

Mininet-HiFi also exploits lightweight virtualization, but adds resource isolation and monitoring to verify that an experiment has run with high fidelity. In this paper, we have demonstrated not only the feasibility of a fidelity-tracking Container-Based Emulator on a single system, but also have shown that these techniques can be used to replicate previously published results.

8. DISCUSSION

We envision a world where every published research work is easily reproducible.[8] The sheer number and scope of the experiments covered in this paper – 19 successfully reproduced – suggest that this future direction for the network systems research community is possible with Container-Based Emulation. CBE, as demonstrated by Mininet-HiFi, meets the goals defined in Section 2 for a reproducible research platform, including functional realism, timing realism, topology flexibility, and easy replication at low cost.

Sometimes, however, CBE can replicate the *result*, but not the exact *experiment*, due to resource constraints. A single server running in real time will inevitably support limited aggregate bandwidth, single-link bandwidth, and numbers of processes. When scaling down an experiment, the right parameters may not be clear, e.g. queue sizes or algorithm tunables. A different result from the scaled-down version may indicate either a configuration error or a not-so-robust result, and the truth may not reveal itself easily.

We intend to build on the work of others to overcome these limits to experiment scale. In the space dimension, ModelNet [54] and DieCast [16] scale emulation to multiple servers. There are research challenges with using this approach in the cloud, such as measuring and ensuring bandwidth between machines with time-varying and placement-dependent network performance. In the time dimension, Time Dilation [29] and SliceTime [56] show methods to slow down the time perceived by applications to run an experiment with effectively larger resources. Perhaps our fidelity measurement techniques could enable a time-dilating CBE to adapt automatically and run any experiment at the minimum slowdown that yields the required level of fidelity.

As a community we seek high-quality results, but our results are rarely reproduced. It is our hope that this paper will spur such a change, by convincing authors to make their next paper a runnable one, built on a CBE, with public results, code, and instructions posted online. If enough authors meet this challenge, the default permissions for network systems papers will change from "read only" to "read, write and execute" – enabling "runnable conference proceedings" where every paper can be independently validated and easily built upon.

[8] Technical societies such as the ACM and IEEE can facilitate reproducible research by providing persistent online repositories for runnable papers.

9. ACKNOWLEDGMENTS

This runnable and largely "crowd-sourced" paper would not have been possible without the dedicated efforts of the students who developed the 18 reproducible experiments in Table 2: Rishita Anubhai, Carl Case, Vaibhav Chidrewar, Christophe Chong, Harshit Chopra, Elliot Conte, Justin Costa-Roberts, MacKenzie Cumings, Jack Dubie, Diego Giovanni Franco, Drew Haven, Benjamin Helsley, John Hiesey, Camille Lamy, Frank Li, Maxine Lim, Joseph Marrama, Omid Mashayekhi, Tom McLaughlin, Eric Muthuri Mibuari, Gary Miguel, Chanh Nguyen, Jitendra Nath Pandey, Anusha Ramesh, David Schneider, Ben Shapero, Angad Singh, Daniel Sommermann, Raman Subramanian, Emin Topalovic, Josh Valdez, Amogh Vasekar, RJ Walsh, Phumchanit Watanaprakornkul, James Whitbeck, Timothy Wong, and Kun Yi.

We are grateful to the anonymous reviewers and our shepherd Sonia Fahmy for their valuable comments and feedback, which helped to improve the final version. We thank the Hedera, Buffer Sizing, and DCTCP authors for sharing their experiment code. This work was partly funded by NSF FIA award CNS-1040190 (NEBULA), the Stanford University Clean Slate Program, and a Hewlett-Packard Fellowship.

References

[1] D. E. Knuth. Literate Programming. *The Computer Journal [Online]*, 27:97–111, 1984.

[2] J. Claerbout. Electronic documents give reproducible research a new

meaning. *Proc. 62nd Ann. Int. Meeting of the Soc. of Exploration Geophysics*, pages 601–604, 1992.
[3] J. B. Buckheit and D. L. Donoho. Wavelab and reproducible research. *Time*, 474:55–81, 1995.
[4] P. Vandewalle, J. Kovacevic, and M. Vetterli. Reproducible research. *Computing in Science Engineering*, pages 5–7, 2008.
[5] Amazon Elastic Compute Cloud. http://aws.amazon.com.
[6] The network simulator - ns-2. http://nsnam.isi.edu/nsnam/.
[7] The ns-3 network simulator. http://www.nsnam.org/.
[8] OPNET modeler. http://www.opnet.com/solutions/network_rd/modeler.html.
[9] Global Environment for Network Innovations. http://www.geni.net/.
[10] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Computer Communication Review*, 33(3):3–12, 2003.
[11] J. DeHart, F. Kuhns, J. Parwatikar, J. Turner, C. Wiseman, and K. Wong. The open network laboratory. In *SIGCSE '06 Technical Symposium on Computer Science Education*, pages 107–111. ACM, 2006.
[12] Emulab - network emulation testbed. http://emulab.net/.
[13] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *NSDI '10*. USENIX, 2010.
[14] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data Center TCP (DCTCP). In *SIGCOMM '10*, pages 63–74. ACM, 2010.
[15] N. Beheshti, Y. Ganjali, R. Rajaduray, D. Blumenthal, and N. McKeown. Buffer sizing in all-optical packet switches. In *Optical Fiber Communication Conference*. Optical Society of America, 2006.
[16] D. Gupta, K. V. Vishwanath, and A. Vahdat. DieCast: Testing distributed systems with an accurate scale model. In *NSDI '08*, pages 407–421. USENIX, 2008.
[17] Open vSwitch: An open virtual switch. http://openvswitch.org/.
[18] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau. Large-scale virtualization in the Emulab network testbed. In *USENIX '08 Annual Technical Conference*, pages 113–128, Berkeley, CA, USA, 2008. USENIX.
[19] M. Pizzonia and M. Rimondini. Netkit: easy emulation of complex networks on inexpensive hardware. In *International Conference on Testbeds and research infrastructures for the development of networks & communities*, TridentCom '08, pages 7:1–7:10, Brussels, Belgium, 2008. ICST.
[20] 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 '08*, pages 72:1–72:6. ACM, 2008.
[21] J. Ahrenholz, C. Danilov, T. Henderson, and J. Kim. CORE: A real-time network emulator. In *Military Communications Conference, MILCOM '08*, pages 1 –7. IEEE, 2008.
[22] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *HotNets '10*, pages 19:1–19:6. ACM, 2010.
[23] S. Soltesz, H. Pötzl, M. Fiuczynski, A. Bavier, and L. Peterson. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. *ACM SIGOPS Operating Systems Review*, 41(3):275–287, 2007.
[24] M. Zec and M. Mikuc. Operating system support for integrated network emulation in IMUNES. In *Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS)*, 2004.
[25] cgroups. http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt.
[26] P. Turner, B. B. Rao, and N. Rao. CPU bandwidth control for CFS. In *Linux Symposium '10*, pages 245–254, 2010.
[27] Handigol, N., Heller, B., Jeyakumar, V., Lantz, B., and McKeown, N. CSTR 2012-02 Mininet performance fidelity benchmarks. http://hci.stanford.edu/cstr/reports/2012-02.pdf.
[28] Linux Trace Toolkit - next generation. http://lttng.org/.
[29] D. Gupta, K. Yocum, M. McNett, A. Snoeren, A. Vahdat, and G. Voelker. To infinity and beyond: time warped network emulation. In *SOSP '05*, pages 1–2. ACM, 2005.
[30] lxc linux containers. http://lxc.sf.net.
[31] K. Ramakrishnan and S. Floyd. A proposal to add explicit congestion notification (ECN) to IP. Technical report, RFC 2481, January 1999.
[32] DCTCP patches. http://www.stanford.edu/~alizade/Site/DCTCP.html.
[33] D. Thaler and C. Hopps. Multipath issues in unicast and multicast next-hop selection. Technical report, RFC 2991, November 2000.
[34] Ripcord-Lite for POX: A simple network controller for OpenFlow-based data centers. https://github.com/brandonheller/riplpox.
[35] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. In *SIGCOMM '04*, pages 281–292. ACM, 2004.
[36] J. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo. NetFPGA – an open platform for gigabit-rate network switching and routing. In *Microelectronic Systems Education, MSE '07*, pages 160–161. IEEE, 2007.
[37] K. Nichols and V. Jacobson. Controlling queue delay. *Communications of the ACM*, 55(7):42–50, 2012.
[38] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda. Less is more: Trading a little bandwidth for ultra-low latency in the data center. In *NSDI '12*. USENIX, 2012.
[39] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley. How hard can it be? designing and implementing a deployable multipath TCP. In *NSDI '12*, pages 29–29. USENIX, 2012.
[40] P. Prakash, A. Dixit, Y. Hu, and R. Kompella. The TCP outcast problem: Exposing unfairness in data center networks. In *NSDI '12*. USENIX, 2012.
[41] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey. Jellyfish: Networking data centers randomly. In *NSDI '12*. USENIX, 2012.
[42] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, and N. Sutin. An argument for increasing TCP's initial congestion window. *SIGCOMM Computer Communication Review*, 40(3):27–33, 2010.
[43] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. Andersen, G. Ganger, G. Gibson, and B. Mueller. Safe and effective fine-grained TCP retransmissions for datacenter communication. *SIGCOMM Computer Communication Review*, 39(4):303–314, 2009.
[44] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu. DCell: a scalable and fault-tolerant network structure for data centers. In *SIGCOMM '08*. ACM, 2008.
[45] N. Dukkipati and N. McKeown. Why flow-completion time is the right metric for congestion control. *SIGCOMM Computer Communication Review*, 36(1):59–62, 2006.
[46] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP congestion control with a misbehaving receiver. *SIGCOMM Computer Communication Review*, 29(5):71–78, 1999.
[47] S. Floyd and V. Jacobson. Random Early Detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.
[48] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving datacenter performance and robustness with multipath TCP. In *SIGCOMM Computer Communication Review*. ACM, 2011.
[49] The OpenFlow switch. http://www.openflow.org.
[50] M. Carson and D. Santay. NIST Net – a Linux-based network emulation tool. *SIGCOMM Computer Communication Review*, 33(3):111–126, 2003.
[51] M. Carbone and L. Rizzo. Dummynet revisited. *SIGCOMM Computer Communication Review*, 40(2):12–20, 2010.
[52] Linux network emulation module. http://www.linuxfoundation.org/collaborate/workgroups/networking/netem.
[53] 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 *SOSP '03*, pages 164–177. ACM, 2003.
[54] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. In *OSDI '02*, pages 271–284. USENIX, 2002.
[55] OpenFlow Virtual Machine Simulation. http://www.openflow.org/wk/index.php/OpenFlowVMS.
[56] E. Weingärtner, F. Schmidt, H. Vom Lehn, T. Heer, and K. Wehrle. Slicetime: A platform for scalable and accurate network emulation. In *NSDI '11*, volume 3. ACM, 2011.
[57] Z. Tan, K. Asanovic, and D. Patterson. An FPGA-based simulator for datacenter networks. In *Exascale Evaluation and Research Techniques Workshop (EXERT '10), at ASPLOS '10*. ACM, 2010.
[58] S. Tripathi, N. Droux, K. Belgaied, and S. Khare. Crossbow virtual wire: network in a box. In *LISA '09*, pages 4–4. USENIX, 2009.