VT-Mininet: Virtual-time-enabled Mininet for Scalable and Accurate Software-Define Network Emulation

Jiaqi Yan
Illinois Institute of Technology
10 West 31st Street
Chicago, IL, United States
jyan31@hawk.iit.edu

Dong Jin
Illinois Institute of Technology
10 West 31st Street
Chicago, IL, United States
dong.jin@iit.edu

ABSTRACT

The advancement of software-defined networking (SDN) technology is highly dependent on the successful transformations from in-house research ideas to real-life products. To enable such transformations, a testbed offering scalable and high fidelity networking environment for testing and evaluating new/existing designs is extremely valuable. Mininet, the most popular SDN emulator by far, is designed to achieve both accuracy and scalability by running unmodified code of network applications in lightweight Linux Containers. However, Mininet cannot guarantee performance fidelity under high workloads, in particular when the number of concurrent active events is more than the number of parallel cores. In this project, we develop a lightweight virtual time system in Linux container and integrate the system with Mininet, so that all the containers have their own virtual clocks rather than using the physical system clock which reflects the serialized execution of multiple containers. With the notion of virtual time, all the containers perceive virtual time as if they run independently and concurrently. As a result, interactions between the containers and the physical system are artificially scaled, making a network appear to be ten times faster from the viewpoint of applications within the containers than it actually is. We also design an adaptive virtual time scheduling subsystem in Mininet, which is responsible to balance the experiment speed and fidelity. Experimental results demonstrate that embedding virtual time into Mininet significantly enhances its performance fidelity, and therefore, results in a useful platform for the SDN community to conduct scalable experiments with high fidelity.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Network Operating Systems; C.2.1 [**Network Architecture and Design**]: Network Communication; I.6.3 [**Simulation and Modeling**]: Application—*Miscellaneous*; D.4.8 [**Operating Systems**]: Performance—*Measurement, Simulation*

SOSR2015, June 17–18, 2015, Santa Clara, CA, USA.

DOI: http://dx.doi.org/10.1145/2774993.2775012.

General Terms

Emulation

Keywords

Virtual Time; Network Emulation; SDN; Mininet

1. INTRODUCTION

Mininet [3] is a network emulator supporting OpenFlow-based Software-defined Networking (SDN), and has been widely adopted by the SDN community. It provides a flexible and cost-efficient experimental platform to develop, test, and evaluate OpenFlow applications. With the lightweight OS-level virtualization technology, Mininet exhibits good scalability (up to 4096 hosts on a commodity laptop [3]) and functional fidelity by running unmodified code of standard Linux network applications over real Linux kernel. However, Mininet fails to provide performance fidelity when the resources required by an emulated network exceed the available CPU or bandwidth on the physical machine. One primary root cause is the serialization of time-stamped events in the containers. If the number of events is more than the number of available cores, not all the events can be processed concurrently like what happens in a real physical testbed. However, the containers and their applications retrieve the timing information from the system clock of the underlying physical machine, which leads to issues of temporal fidelity because a container's clock still advances if even it is not running (e.g., idle, waiting, suspended). Such errors would greatly affect the performance fidelity of Mininet, particularly for emulating high workload network scenarios.

We first demonstrate the limitations of Mininet with three motivation examples, and then present our approach of a lightweight virtual time system in Mininet, named VT-Mininet, to address the performance fidelity issues. We believe that VT-Mininet will be very useful to the SDN community, especially because of the extremely wide usage of Mininet, as a platform to perform scalable emulation experiments with high fidelity.

1.1 Exploring Limitations of Mininet

We first explore the limitations of Mininet (version 2.1.0, also named Mininet-Hifi) through three sets of experiment running on a Linux box (Ubuntu 14.04.1) with one Intel i7-4790 CPU and 12 GB RAM. Each experiment was independently repeated for 10 times. The experimental results demonstrate that Mininet is unable to provide sufficient performance fidelity, when the aggregated resource demand re-

Figure 1: **A Switch Chain Network Topology for Motivation Experiment 1 and 2.**

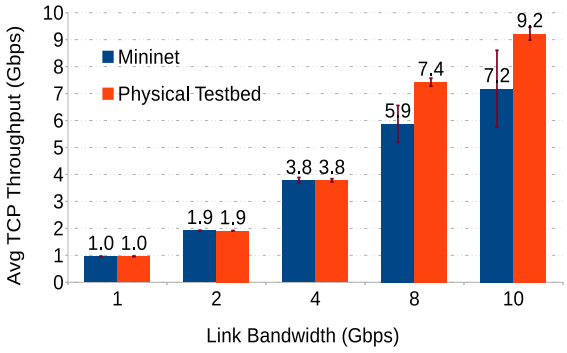

Figure 2: **Motivation Experiment 1: Single TCP Flow Throughput with Different Bandwidth**

quired by an emulation experiment exceeds the host machine's resource capacity.

Performance Fidelity. We built a 40-Open-vSwitch chain topology in Mininet (see Figure 1) connected by 10 Gbps links with 10 μs latency (constructed by `TCLink`). Meanwhile, we also built a simple physical testbed with two hosts connected by a 10 Gbps link using Intel 82599-based network interface cards. We used `tc` to vary the bandwidth of the physical link with the latency set to be the corresponding round trip time (RTT) measured in Mininet. The throughputs measured by `iperf3` [1] on both testbeds are plotted in Figure 2 for comparison. We can see that Mininet was capable to accurately emulate the single flow when the link bandwidth was no greater than 4 Gbps, but the throughputs were significantly smaller than the physical testbed results when the bandwidth increased to 8 Gbps and above.

Scalability. With the same switch chain topology, we fixed the link bandwidth to 4 Gbps with 50 μs delay. This time, we varied the number of switches from 10 to 80, and measured the single flow throughputs with `iperf3`. Results are plotted in Figure 3. We observe that the throughput plummetted away from the desired behavior (i.e., close to the line rate 4 Gbps) as the number of switches grew, e.g., 44% drop for 60 switches and 67% drop for 80 switches.

Time Overlapping Events. Container-based emulators offer functional fidelity through direct code execution, but often encounter temporal errors when the number of concurrent events to process is larger than the available processors. To demonstrate this issue, we created a network topology (see Figure 4) with all the links set to 1 Gbps with 100 μs latency. Among the 50 hosts, we selected 5 pairs of hosts: `(h1, h10), (h11, h20), h(21, h30), h(31, h40), (h41, h50)`. The experiments had two phases. We first generated a single TCP flow between `h1` and `h10`. After 50 seconds, we started to transmit other 4 TCP flows. Each flow was repeatedly transmitted with one transmission lasting for 5 seconds. Figure 5 depicts the throughputs of all five flows over one run. In phase 1, Mininet accurately emulated the throughput (close to 1 Gbps) of the single TCP flow. In phase 2, the

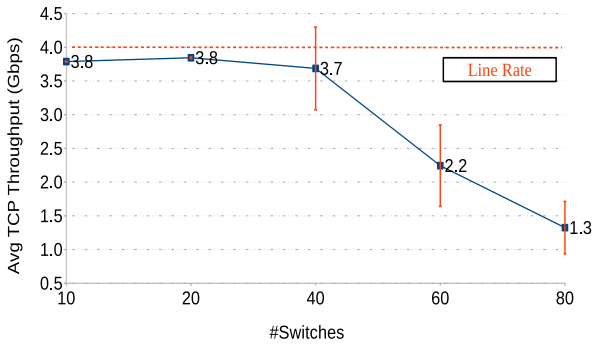

Figure 3: **Motivation Experiment 2: Single TCP Flow Throughput with Varying Number of Switches in the Chain**

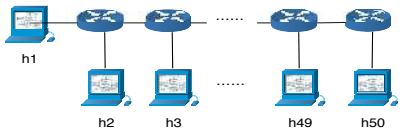

Figure 4: **Network Topology for Motivation Experiment 3**

ideal behavior should be around 1 Gbps throughput for every flow, since the five flows were designed to have no overlapped communication paths. However, the emulated throughputs were far below the line rate with large disturbances. We also plot the congestion window (cwnd) size of all five flows during one transmission in phrase 2, as shown in Figure 6. The increments of cwnd size significantly vary among all the five flows, in particular, 4 Mbytes for flows between `h11-h20, h21-h30, h31-h40` and 132 Kbytes for the other two. Apparently, there were insufficient resources to process all the flows, and the CPU time was unfairly distributed among the emulated hosts. When an emulated host did not get its CPU time slice, its time still advanced. It is because all the containers used the system clock in spite of its status (idle, sleep, suspended, etc.). Therefore, some hosts perceived incorrectly longer RTTs, which led to low throughputs. The root cause of the *temporal fidelity* issues in Mininet is that the execution of containers is in serial (scheduled by the OS), and the time-stamped events occurred in the containers reflect that serialization.

1.2 Improving Mininet with Virtual Time

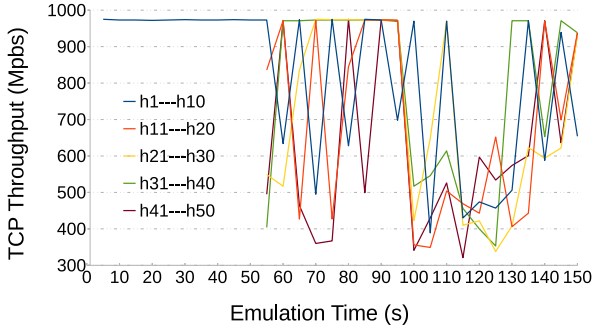

Figure 5: **Motivation Experiment 3: Throughputs of 5 Concurrent TCP Flows**

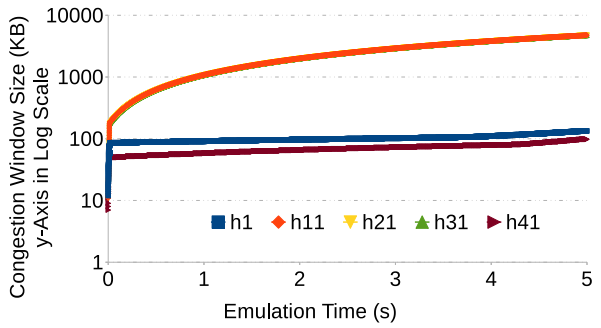

Figure 6: **TCP Congestion Window Traces, Five Concurrent TCP Flows**

Virtual time sheds the light on the temporal fidelity issues. The key insight is to trade time with system resources in the way of precisely scaling the system's capacity to match behaviors of the target networks, which typically exceed the available physical resources. The idea of virtual time has been explored in the form of time-dilation-based [11] and virtual-machine-scheduling-based [21, 22] designs and has been applied to various virtualization techniques including Xen [9, 10], OpenVZ [15], and Linux Container [16].

In this work, we take a time-dilation-based approach to build a lightweight middleware embedding Linux containers in virtual time, and integrate it with Mininet. The time dilation factor (TDF) is defined as the ratio between the rate at which time passes in wall clock to the emulated host's perception of time. A TDF of 10 means that for every ten seconds of real time, all applications running in a time-dilated emulated host perceive the time advancement as one second. This way, the interactions between containers and physical devices can be artificially scaled, and a 100 Mbps link now appears to be a 1 Gbps link from the emulated host's viewpoint.

To the best of our knowledge, we are the first to apply virtual time in the context of SDN. Our virtual time system is lightweight and consists of a small set of modifications to the Linux kernel. It transparently provides virtual time to network applications (no code changes are needed) in Mininet, while returns the ordinary system time to other processes. We also designed an adaptive time dilation scheme to optimize the performance tradeoff between execution speed and experiment fidelity. Experimental results indicate that VT-Mininet is capable to ensure performance fidelity for emulating much larger networks that Mininet is incapable to emulate due to resource shortages, typically by the factor of TDF.

2. RELATED WORK

2.1 Virtual Time System

Virtual time has been investigated to improve the scalability and fidelity of virtual-machine-based network emulation. There are two main approaches to develop virtual time systems. The first approach is based on time dilation, a technique to uniformly scale the virtual machine's perception of time by a specified factor. It was first introduced by Gupta et al. [11], and adopted to various types of virtualization techniques and integrated with a handful of network emulators. Examples include DieCast [10], SVEET [6], NETbalance [8], TimeJails [9, 18] and TimeKeeper [16]. The second approach focuses on synchronized virtual time by modifying the hypervisor scheduling mechanism. Hybrid testing systems that integrate network emulation and simulation have adopted this approach. For example, S3F [15] integrates an OpenVZ-based virtual time system [22] with a parallel discrete-event network simulator by virtual time. SliceTime [21] integrates ns-3 [13] with Xen to build a scalable and accurate network testbed.

Our approach is technically closest to TimeKeeper [16] through direct kernel modification of time-related system calls in the Linux kernel to build a lightweight virtual time system. The differences are (1) we are the first to apply virtual time in the context of SDN emulation, (2) VT-Mininet has a wider coverage of system calls interacting in virtual time, and (3) VT-Mininet has an adaptive time dilation scheduling algorithm to well balance speed and fidelity for emulation experiments.

2.2 SDN Emulation Testbeds

OpenFlow [19] is the first standard communications interface defined between the control and forwarding planes of an SDN architecture. Examples of OpenFlow-based SDN emulation testbeds include MiniNet [17], MiniNet-HiFi [12], EstiNet [20], ns-3 [13] and S3F [14]. Mininet is the most widely used SDN emulation testbed at present, which uses process-based virtualization technique to provide a lightweight and inexpensive network emulation testbed. Ns-3 [13] has an OpenFlow simulation model and also offers a realistic OpenFlow environment through its generic emulation capability. S3F supports scalable simulation and emulation of OpenFlow-based SDN through a hybrid platform that integrates a parallel network simulator and a virtual-time-enabled OpenVZ-based network emulator [5].

3. DESIGN AND IMPLEMENTATION

3.1 Mininet Network Emulation

Mininet uses Linux container [2], a lightweight OS-level virtualization to achieve scalable SDN emulation. A container allows a group of processes, including the process of the virtual hosts and the application processes running inside the container, to have an independent view of system resources including process ID, file system and network interfaces, but still share the kernel with other containers. Mininet supports a few types of OpenFlow-enabled switches, such as Open vSwitch [4] operating in kernel space for efficiency. Those switches are connected among themselves via virtual links and connected to containers via virtual interfaces to form the emulated network topology, as shown in Figure 7. Mininet-Hifi supports high fidelity network emulation via resource provisioning and performance isolation. For example, to ensure no virtual hosts are starved by the OS process scheduler, Mininet-Hifi uses `cgroup` for provisioning CPU time slice among emulated hosts; to emulate links with accurate bandwidth and latency, Mininet-Hifi uses `tc` to configure links and schedule packets across network namespaces. However, Mininet still faces performance fidelity issues as discussed in Section 1.1.

3.2 Virtual-Time-Enabled Mininet

To improve the performance fidelity, we take a time-dilation-base approach to develop a lightweight virtual time system

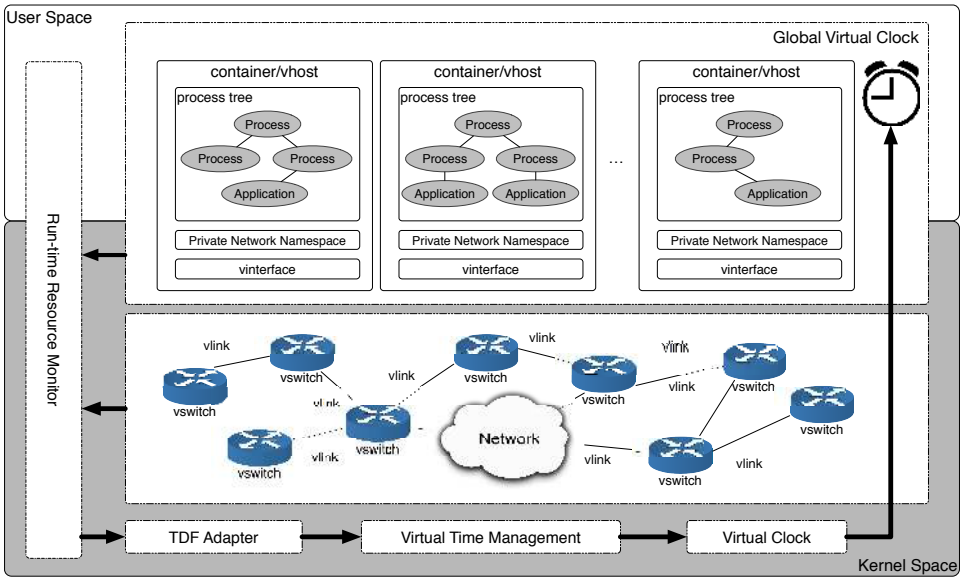

Figure 7: **Architecture of VT-Mininet**

in Linux container, and integrate it to Mininet. The design architecture of VT-Mininet is depicted in Figure 7. The virtual time system is a thin middleware that consists of two major components: the virtual time manager and the adaptive virtual time scheduler. The virtual time manager resides in the kernel and is responsible for computing and maintaining virtual time, according to a given TDF for all the containers. It can offer per-container virtual time or the global virtual time to Mininet. The per-container virtual time is useful to support synchronized emulation and enables the integration with a network simulator. In this paper, we use global virtual time for all the experiments since no simulation is involved. The key insight of virtual time is to trade time with emulation resources. A fixed TDF, typically large enough to guarantee fidelity, can be overestimated for the dynamic emulation workload, and thus lead to unnecessarily long execution time. The adaptive virtual time scheduler is designed to tackle this problem. As shown in Figure 7, the run-time resource monitor collects system information such as CPU utilization based on which the TDF adaptor tunes the value of TDF with a heuristic threshold-based algorithm. Our virtual time system implementation includes a small set of modifications in the kernel (8 files, less than 500 lines of code) and the Mininet source code (2 files, less than 80 lines of code).[1]

3.2.1 A Linux-container-based Virtual Time System

We make a small set of modifications to the Linux kernel that enables the creation of Linux container with virtual clock. We add 4 new fields (declared in Algorithm 1, less than 32 bytes in total) in the `task_struct` struct type so that a process can have its own perception of time. We also added a private function `do_dilatetimeofday` in Linux's timekeeping subsystem to track the dilated virtual time based on the elapsed physical time and TDF, as shown in Algorithm 1. Based on a process's `virtual_start_nsec`, the system determines the type of time to return, either the physical system clock time or the virtual time. To process virtual time inquiries, the time passed since the previous call to `do_dilatetimeofday` is calculated and precisely scaled with TDF. To enable virtual time support for a wide range of timing-related system calls, we extensively traced the routines in Linux's subsystems that request timing information (such as `getnstimeofday`, `ktime_get`, `ktime_get_ts`, etc.), and modified them to properly invoke `do_dilatetimeofday`.

To enable the virtual time perception to processes running in Mininet, we make the following process-related kernel modifications.

- `virtualtimeunshare` is essentially the `unshare` system call, which Mininet invokes to create `Nodes`, but with the additional time dilation parameters. It creates a new process with a TDF in a different namespace of its parent process.
- `settimedilaitonfactor` exposes an interface for Mininet to change the TDF of a process. In Mininet, since a command executed in a host is equivalent to a shell command executed by `bash`, adjustment of a process' TDF should also include changing the TDF of its parent process (e.g., host's `bash`). This operation occasionally would lead to tracing back to the root of the process tree, especially in the case of dynamic TDF adjustment.

Since the Linux kernel code has several ways to retrieve time information and our implementation did not cover all of them. In this paper, we focus on capturing all the related system calls and kernel routings to support virtual time in Mininet. One particular case related is the traffic control command `tc`, used by Mininet to emulate resource isolated links. For example, if we set TDF as 8 in Mininet and tune `tc` to set up a rate limited 100Mbps link using Hierarchic Token Bucket (HTB) `qdisc`, the resulting link bandwidth will be approximately 800Mbps tested by time dilated program `iperf3`. The issue is that `tc` does not reference the Linux kernel's time. Therefore, `tc` bypasses to our virtual time system. One way to solve this problem is to modify the network scheduling code in kernel to provide `tc` with a dilated traffic rate. In the earlier example with TDF set to 8, the experiment will run 8 times slower than the real time, and we can configure `tc`'s rate limit as $rate/TDF = 12.5$ Mbps to emulate a 100 Mbps link. As for now, we tailored

[1] `https://github.com/littlepretty/VirtualTimeForMininet`

Algorithm 1 Time Dilation Algorithm

```
1: New variables in task_struct
2: Let p denote a process created in Mininet
3: p.virtual_start_nsec        /* the starting time that a process detaches from the system clock and uses the virtual
   time, initialized by getnstimeofday */
4: p.physical_past_nsec        /* how much physical time has elapsed since the previous time inquiry */
5: p.virtual_past_nsec         /* how much virtual time has elapsed since the previous time inquiry */
6: p.dilation         /* a process's time dilation factor */
7:
8: function DO_DILATETIMEOFDAY(struct timespec *ts)
9:     if p.virtual_start_nsec > 0 and p.dilation > 0 then
10:        now = timespec_to_ns(ts)
11:        physical_past_nsec = now - p.virtual_start_nsec
12:        /* virtual time computation */
13:        virtual_past_nsec = (physical_past_nsec - p.physical_past_nsec)/p.dilation + p.virtual_past_nsec
14:        dilated_now = virtual_past_nsec + p.virtual_start_nsec
15:        dilated_ts = ns_to_timespec(dilated_now)
16:        ts→tv_sec = dilated_ts.tv_sec
17:        ts→tv_nsec = dilated_ts.tv_nsec
18:        p.physical_past_nsec = physical_past_nsec       /* update process's physical time */
19:        p.virtual_past_nsec = virtual_past_nsec       /* update process's virtual time */
20:    end if
21: end function
```

Hierarchic Token Bucket (HTB), the default `qdisc` used by Mininet, in `tc`.

3.2.2 Virtual Time Integration with Mininet

We design the virtual time system to allow ease of integration with Mininet as well as other linux-container-based applications. Mininet calls `virtualtimeunshare` instead of `unshare` to create emulated hosts. This way, a global TDF can be set among all hosts. The modifications are mainly in `mnexec`, a backend `C` program in Mininet.

- When Mininet creates `Node`s (hosts, switches and controllers are inherited from `Node`) with `-n` option, users can pass in an integer value of TDF as an argument of `virtualtimeunshare`.
- We also provide the ability to dynamically adjust the TDF for every emulated host during runtime. To do that, we added a new option `-t` in `mnexec` to invoke the aforementioned system call `settimedilaitonfactor` to do the actual TDF adjustment.

Network applications running inside containers (i.e., `iperf3` and `ping`) should also use virtual time. We do not need to modify the application code because they are running as child processes of Mininet's hosts. A child process always copies its parent's `task_struct` when it is forked including the same TDF and `virtual_start_nsec` values. Although `virtual_start_nsec` does not present the virtual start time of the child process, our algorithm is designed to work with relative values. When applications inquire about the system time, the modified kernel knows that they are using virtual clock and return them virtual clock time instead of system clock time.

3.3 Adaptive TDF Scheduling

To optimize the performance of the VT-Mininet, we developed an adaptive TDF scheduler through two python modules `MininetMonitor` and `DilationAdaptor` (refer to Runtime Resource Monitor and TDF Adaptor in Figure 7) to accelerate the experiment speed while preserving the performance fidelity.

We added a python module `MininetMonitor` to monitor the CPU usage of the entire emulation system, consisting of a group of processes including the Mininet emulator, Open vSwitch module, and emulated nodes. `MininetMonitor` utilizes the `ps` command to collect the group's aggregate CPU percentage and periodically computes and passes the average CPU load statistics to `DilationAdaptor`. The core of `DilationAdaptor` is an adaptive algorithm to calculate an appropriate TDF. We currently take a similar threshold-based algorithmic approach as described in TimeJails [7]. We will leave the investigation of other effective control indicators, such as average process waiting time, other than CPU utilization as the future work.

4. EXPERIMENTAL EVALUATION

In this section, we evaluate VT-Mininet by repeating the three experiments in Section 1.1 using the VT-Mininet with TDF set to 4 on the same physical machine, and demonstrate how the virtual time can help to improve Mininet's performance. We also report the system overhead.

Performance Fidelity. First, we conducted the single TCP flow throughput measurement over various link bandwidths on the 40-Open-vSwitch chain topology in Mininet (see Figure 1). As shown in Figure 8, with virtual time (TDF = 4), Mininet is capable to accurately emulate the high-bandwidth TCP flows, and the results closely match the physical testbed measurements even at 10 Gbps, while the original Mininet experiences significant drops. It is because with virtual time, it appears to the containers that the system is now sufficiently fast (approximately 4 times faster) to correctly process the application activities in a timely manner.

Scalability. We also want to explore how virtual time improves the scale of networks that Mininet can emulate without losing fidelity. We re-used the same switch chain topology in Figure 1 with 4 Gbps link bandwidth. Figure 9

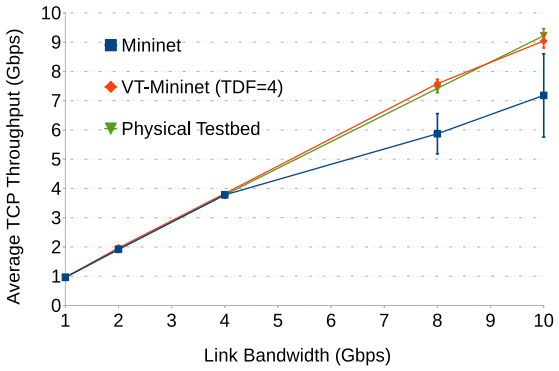

Figure 8: **Single TCP Flow Throughput with Different Bandwidth**

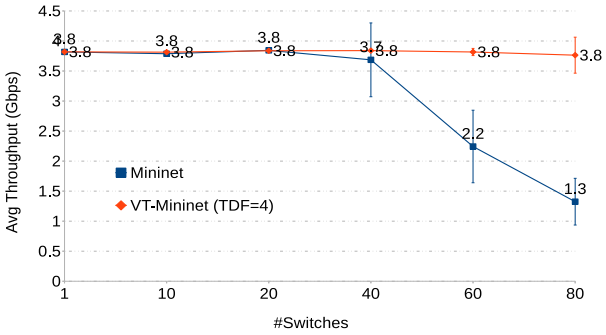

Figure 9: **Single TCP Flow Throughput with Varying Number of Switches in the Chain**

shows that using VT-Mininet, TCP throughputs remained near 3.8 Gbps with very small standard derivations in all the experiments. It is clear that virtual time helps to scale up the emulation. VT-Mininet can emulate 80 switches with 4 Gbps links and still generate the accurate throughputs, rather than being saturated at 20 switches without virtual time.

Longer execution time is the tradeoff for the fidelity and scalability improvement. We also recorded the emulation running time, and observe that the time is approximately proportional to TDF. In this case, TDF was set to 4, and VT-Mininet's execution time was about 4 times longer than the original Mininet, e.g., 240 second in Mininet and 980 seconds in VT-Mininet in the 80-switch case.

Time Overlapping Events. We then repeated the two-phase experiments (single flow in phase 1 and five concurrent flows in phase 2) with VT-Mininet on the network topology shown in Figure 4. Figure 10 plots the flow throughputs in both phases. Compared with the results in Figure 5 in phase 2, the throughput of all five flows were very unstable without virtual time because of the heavy resource contention and the underlying physical system being incapable to support emulation of five concurrent flows. In contrast, with virtual time, all five flows have stable throughputs around 1 Gbps. Further exploration of the congestion window shows that all senders can quickly reach 4-MBytes congestion windows in VT-Mininet. This is the desired behaviors because the five flows did not share any communication paths and they ought to achieve the close-to-line-rate throughputs in physical world applications.

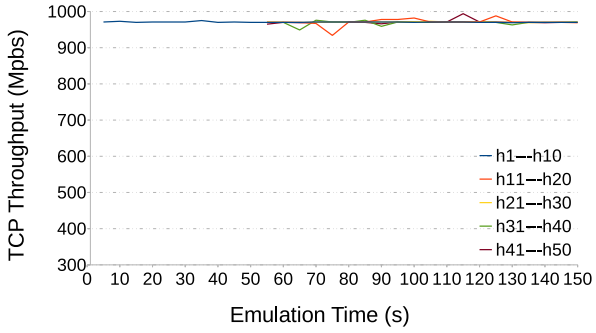

Figure 10: **Throughputs of 5 Concurrent TCP Flows**

System Overhead. Our virtual time system brings overhead with the following two reasons: (1) the computation cost in Algorithm 1 and (2) the pauses of emulation when changing containers' TDFs. We measured both types of overhead and report the results in this section.

First, we invoked both non-dilated and dilated `gettimeofday` 10,000,000 times from a user space application. The average overhead for one dilated `gettimeofday` is 0.013 microseconds. We then used `strace` to count the number of invocations for `gettimeofday` in a 60-second `iperf3` run on both the server and the client sides. The total overhead is 18,145 microseconds after tracing 1,397,829 calls, which is about 0.03% of the 60-second experiment. Actually, `iperf3` intensively invokes `gettimeofday`, because its timer is designed to exhaustively inquiry OS time. The overhead amount will be even less for many other network applications. We also repeatedly changed a process's TDF 10,000,000 times using another test program. The average pause time was 0.063 microseconds, which is reasonably small. Since the number of TDF changes issued by the current adaptive TDF scheduling algorithm is a few orders of magnitude less than the number of calls to `gettimeofday`, the overhead is negligible. We will conduct further analysis of other adaptive TDF scheduling algorithms in the future.

5. CONCLUSION AND FUTURE WORKS

We present VT-Mininet, a virtual-time-embedded SDN network emulator, enhancing Mininet's performance fidelity. We took a time-dilation-based approach to build the virtual time system that are lightweight and transparent to applications. Experimental results show the clear improvement over Mininet for high workload network experiments. We will conduct more extensive analysis of VT-Mininet with real-world SDN applications. We will also investigate other control algorithms to improve the adaptive TDF scheduler for performance improvement in the future.

6. ACKNOWLEDGMENTS

This material is based upon work supported by the Maryland Procurement Office under Contract No. H98230-14-C-0141[2]. The authors are grateful to Jeremy Lamps for sharing the code base of TimeKeeper [16].

[2]**Disclaimer.** Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Maryland Procurement Office.

7. REFERENCES

[1] iperf3. http://software.es.net/iperf.

[2] Linux containers. https://linuxcontainers.org.

[3] Mininet: An instant virtual network on your laptop (or other PC). http://mininet.org/.

[4] Open vSwitch. http://openvswitch.org.

[5] S3F/S3FNet. https://s3f.iti.illinois.edu/.

[6] M. Erazo, Y. Li, and J. Liu. Sveet! a scalable virtualized evaluation environment for tcp. In *Proceedings of the 2009 Testbeds and Research Infrastructures for the Development of Networks Communities and Workshops*, pages 1–10, 2009.

[7] A. Grau, K. Herrmann, and K. Rothermel. Efficient and scalable network emulation using adaptive virtual time. In *Proceedings of the 18th International Conference on Computer Communications and Networks*, pages 1–6, 2009.

[8] A. Grau, K. Herrmann, and K. Rothermel. Netbalance: Reducing the runtime of network emulation using live migration. In *Proceedings of the 20th International Conference on Computer Communications and Networks*, pages 1–6, 2011.

[9] A. Grau, S. Maier, K. Herrmann, and K. Rothermel. Time jails: A hybrid approach to scalable network emulation. In *Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation*, pages 7–14, 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 Transactions on Computer Systems*, 29(2):1–48, 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 *Proceedings of the 20th ACM Symposium on Operating Systems Principles*, pages 1–2, 2005.

[12] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. Reproducible network experiments using container-based emulation. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, pages 253–264, 2012.

[13] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena. Network simulations with the ns-3 simulator. *SIGCOMM Demonstration*, 15:17, 2008.

[14] D. Jin and D. M. Nicol. Parallel simulation of software defined networks. In *Proceedings of the 2013 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 91–102, 2013.

[15] D. Jin, Y. Zheng, H. Zhu, D. M. Nicol, and L. Winterrowd. Virtual time integration of emulation and parallel simulation. In *Proceedings of the 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation*, pages 201–210, 2012.

[16] J. Lamps, D. M. Nicol, and M. Caesar. Timekeeper: A lightweight virtual time system for linux. In *Proceedings of the 2nd ACM SIGSIM/PADS Conference on Principles of Advanced Discrete Simulation*, pages 179–186, 2014.

[17] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 1–6, 2010.

[18] S. Maier, A. Grau, H. Weinschrott, and K. Rothermel. Scalable network emulation: A comparison of virtual routing and virtual machines. In *Proceedings of 12th IEEE Symposium on Computers and Communications*, pages 395–402, 2007.

[19] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[20] S.-Y. Wang, C.-L. Chou, and C.-M. Yang. Estinet openflow network simulator and emulator. *Communications Magazine, IEEE*, 51(9):110–117, 2013.

[21] E. Weingärtner, F. Schmidt, H. V. Lehn, T. Heer, and K. Wehrle. Slicetime: A platform for scalable and accurate network emulation. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, pages 253–266, 2011.

[22] Y. Zheng and D. M. Nicol. A virtual time system for openvz-based network emulations. In *Proceedings of the 2011 IEEE Workshop on Principles of Advanced and Distributed Simulation*, pages 1–10, 2011.