*Proceedings of the 2017 Winter Simulation Conference*
*W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, eds.*

# A HARDWARE-IN-THE-LOOP EMULATION TESTBED FOR HIGH FIDELITY AND REPRODUCIBLE NETWORK EXPERIMENTS

Xiaoliang Wu
Qi Yang
Xin Liu
Dong Jin

Illinois Institute of Technology
10 West 31st Street
Chicago, IL, USA

Cheol Won Lee

National Security Research Institute
1559, Yuseong-daero, Yuseong-gu
Daejeon, SOUTH KOREA

## ABSTRACT

The transformation of innovative research ideas to production systems is highly dependent on the capability of performing realistic and reproducible network experiments. In this work, we present a network testbed consisting of container-based network emulation and physical devices to advocate high fidelity and reproducible networking experiments. The testbed integrates network emulators (Mininet), a distributed control environment (ONOS), and physical switches (Pica8). The testbed (1) offers functional fidelity through unmodified code execution in emulated networks, (2) supports large-scale network experiments using lightweight OS-level virtualization techniques and capable of running across distributed physical machines, (3) provides the topology flexibility, and (4) enhances the repeatability and reproducibility of network experiments. We validate the testbed fidelity through extensive experiments under different network conditions (e.g., varying topology and traffic pattern). We also use the testbed to reproduce key results from published network experiments, such as Hedera, a scalable and adaptive network traffic flow scheduling system.

## 1 INTRODUCTION

Modern data centers are built on the tremendous success of cloud computing and storage virtualization technologies. With software-defined networking (SDN), the communication networks in the data centers are increasingly being transformed to function more like the virtualized versions of compute and storage today. SDN seeks to simplify network management by decoupling the control logic from the underlying network infrastructure, and offers online and direct network programmability to enable innovation. There is extensive ongoing academic and industrial research to make SDN proliferate more networks beyond data centers, such as wide area networks, industrial control systems, and wireless networks. It is therefore critical to have a testing and evaluation framework to enable researchers to conduct high fidelity analysis and evaluation of their research ideas before the real system deployment.

An ideal testbed ought to offer desired fidelity, scalability, flexibility and reproducibility for network experiments. Fidelity stands for the ability to provide accurate experimental results. Scalability concerns about the scale of network experiments that a testbed can afford to execute. The testbed should also offer flexible and low-cost ways for users to configure network experiments to explore different topologies and architecture designs. Last but not the least, it is crucial to make network research ideas reproducible, a standard for research papers in many other fields (e.g., cell biology), but unfortunately not in networking research.

Currently, there are three common types of SDN testbeds, i.e., simulation, emulation, and physical testbeds, and each type has different advantages and disadvantages in terms of the aforementioned properties. Several widely-used physical network testbeds start to provide SDN experimentation capabilities to users, including DeterLab (Mikovic et al. 2017), GENI (Berman et al. 2014), OFelia (Suñé et al. 2014), ESNet (DOE 2017), Felix (Carrozzo et al. 2014), and CloudLab (Ricci and Eide 2014). They offer high fidelity, but are often technically challenging and economically infeasible to perform large-scale experiments, and it is also hard to reproduce experimental results. Simulation testbeds with SDN capability, such as OpenNet (Chan et al. 2014), EstiNet (Wang, Chou, and Yang 2013), NS-3 (Jurkiewicz 2013), S3FNet (Jin and Nicol 2015), fs-sdn (Sommers 2013), and OMNet++ (OMNet++ 2013), on the other hand offer good scalability and flexibility to set up network experiments, but have relatively low fidelity due to model abstraction and simplification. Virtual-machine-based emulation testbeds, such as Mininet (Handigol et al. 2012), offer a balanced solution: they allow unmodified code execution for high functional fidelity, and the lightweight OS-level virtualization also offers reasonably good scalability (e.g., supporting 1000+ containers in a commodity server). However, virtual machines share the underlying physical resources, which may violate timing realism due to resource contention, scheduling serialization, and background system load. To improve the realism of virtual machine based emulation, VT-Mininet (Yan and Jin 2015) patched emulation with virtual time support, so that one can slow down the overloaded experiments with still accurate timing to alleviate the infidelity caused by resource multiplexing.

In this paper, we develop an SDN testbed that efficiently integrates the multiple Mininet emulators and physical devices with the ONOS distributed control environment (Berde et al. 2014). The testbed also supports distributed emulation over a cluster of physical machines to enable large-scale experiments. For certain critical or heavily loaded network devices and end-hosts in the target network, we can replace the emulated nodes with hardware devices (e.g., Pica8 SDN switches, embedded Linux devices) to enhance fidelity. We also build the data channel, cluster channel, and control channel among the various components to handle network traffic, SDN controller traffic, and experiment control messages respectively. We then conduct extensive experiments to evaluate various aspects of our testbed, including the performance improvements contributed by the lightweight container-based emulation, the distributed emulation setup, and the seamless physical device integration. More testbed validation results, such as duplexity, bandwidth sharing, bottleneck link performance, are reported in (Yang 2017). Finally, we implement and reproduce an existing network research work, Hedera (Al-Fares, Radhakrishnan, Raghavan, Huang, and Vahdat 2010), using our testbed. Hedera is a dynamic network traffic flow scheduling scheme for data center networks. We demonstrate that Hedera can achieve higher aggregate bandwidth than the Equal Cost Multipath (ECMP) scheduling (Hopps 2000) in fat-tree topology based data center networks, and the results match well with the physical testbed results in the original paper.

The remainder of the paper is organized as follows. Section 2 describes the testbed design. Section 3 presents the performance evaluation. Section 4 demonstrates the testbed usability with a case study of a data center scheduling scheme. Section 5 concludes the paper with future work.

## 2 DESIGN

We develop an SDN testbed with the goal of achieving high fidelity while maintaining scalability, flexibility and reproducibility. Our testbed efficiently integrates container-based emulation and physical devices, and is capable of executing network experiments across distributed physical machines.

### 2.1 Testbed Overview

Figure 1 presents the testbed architecture. The control layer consists of the Operating Network Operating System (ONOS) controller (Berde et al. 2014) over a cluster of servers (i.e., 2U 2.6 GHz Dual Intel Xeon machines with 64 GB RAM, two Gigabit Ethernet ports). ONOS supports a distributed controller environment, which significantly increases the size of network experiments that one can emulate. ONOS

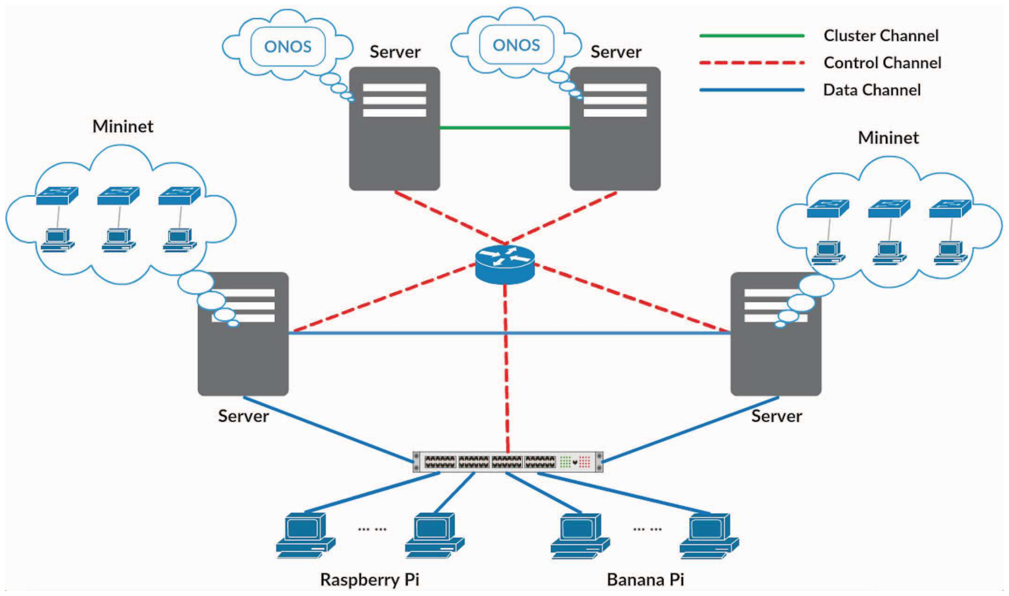


Figure 1: Testbed architecture.

also provides many unique features to enhance the usability of the testbed, such as host-to-host intents, multi-path forwarding, and traffic statistic monitoring. ONOS also offers a GUI for network experiment customization and visualization.

The network layer is built on the container-based network emulation, which enables high fidelity analysis by allowing real networking applications to run in the network emulator and interact with the controller. We use Mininet (Handigol et al. 2012) as the network emulator. It contains software switches that emulate the function of real SDN switches as well as virtual hosts with their own namespaces that can run real processes. Our testbed provides flexible configuration and direct network programmability as an inherent advantage by adopting Mininet. To enhance the testbed scalability, we enable the distributed emulation feature in our testbed by connecting multiple instances of Mininet on different physical machines (i.e., commodity servers with 3.4GHz Intel i7 CPU, 16G RAM, one Gigabit Ethernet port and one wireless network interface). In addition, the network layer also supports physical hardware integration to enhance the testbed fidelity. We have integrated physical switches like Pica8 P-3297 as well as physical end-hosts like Raspberry Pi and Banana Pi. The testbed can run network experiments in a hybrid mode concurrently supporting both virtual and physical devices.

There are three communication channels in our testbed at experiment run time as shown in Figure 1. The cluster channel is responsible for the communication between the controller and the applications, the data channel is responsible for transmitting the network traffic among network devices, and the control channel enables the flexible configuration of network components and allows researchers to tightly control and command network experiments.

## 2.2 Testbed Components

### 2.2.1 Container-based Network Emulator

Container-based virtualization is a lightweight operating-system-level virtualization technology. A container is a Linux process with its own namespace. In the container-based network emulator, each host is implemented as a container with its own virtual network interface(s) and allows execution of real networking applications to improve fidelity. We integrate such a container-based network emulator, Mininet (Handigol et al. 2012), in our testbed. Mininet incorporates software switches (Open vSwitch) as forwarding devices and uses virtual Ethernet (veth) to emulate network links. To enhance flexibility, Mininet provides both APIs and an interactive command-line interface for users to configure the network and to run emulation experiments.

#### 2.2.2 Distributed Emulation

The fact that containers share the same underlying physical infrastructure confines the ability to accurately emulate large-scale networks due to resource contention. Rather than using a super node with a massive number of cores and huge memories, we investigate distributed network emulation. We manage to connect multiple Mininet instances on different physical machines, each Mininet is responsible for constructing a portion of the network. There are two ways to connect the Mininet instances. First, we can assign physical ports to the OVS switches in Mininet and then link those ports with network cables. Second, we can use generic routing encapsulation (GRE) tunnel to connect the Mininet instances on different machines. The first method is fast and the number of interconnected links is limited by the number of available physical ports. The second method has some performance drawback compared with the physical connection but can support a large number of interconnected links among the distributed network emulators.

#### 2.2.3 Physical Devices

The virtual devices in the network emulator can execute unmodified applications and process real network packets. However, those virtual components have to compete for physical resources (e.g., CPU) during runtime, so the timing realism may be violated. One way to improve test fidelity is to replace the virtual devices with physical devices at certain critical locations of a network experiment. Physical devices use the actual network interfaces and cables to transmit packets rather than using the virtual interfaces, and they do not need to compete for system resources as the emulators do. Therefore, adding physical devices can improve functional realism, timing realism, and traffic realism. Specifically, we manage to integrate Pica 8 white box switches in our testbed. Those switches pre-load an operating system and support the OpenFlow protocol so that they can seamlessly talk to the ONOS controller and other virtual switches in Mininet. We can also slice one physical switch into several sub-switches to enhance the flexibility of network experiment configuration.

### 2.3 Experiment Configuration

We utilize the additional control channel (see Figure 1) to execute customized scripts to configure the network experiments, including topology, switches, hosts, and controller applications. The procedures include (1) assigning IP addresses and port numbers to the controllers, (2) slicing the Pica8 switches into numerous smaller switches using `ovs` commands, (3) connecting all the switches to controllers, (4) generating a network topology with virtual switches and hosts, and connecting the virtual devices to the controllers, (5) allocating the physical ports to the virtual switches in order to connect the virtual switches on different physical devices. During experiment execution, we can even log into the virtual/physical hosts via the additional control channel to run shell commands, such as starting or stopping a network application, collecting measurement data, etc.

## 3 EXPERIMENTAL EVALUATION

We have performed extensive experiments to evaluate and validate our testbed. In this section, we select three sets of experimental results to demonstrate (1) how the distributed emulation improves scalability as well as fidelity when there is a resource contention (§3.1), (2) how the hardware-in-the-loop setup enhances the fidelity of experiments (§3.2), and (3) how the physical device placement affects the testbed performance (§3.3). More testing results, such as duplexity, bandwidth sharing, bottleneck link performance are reported and discussed in (Yang 2017).

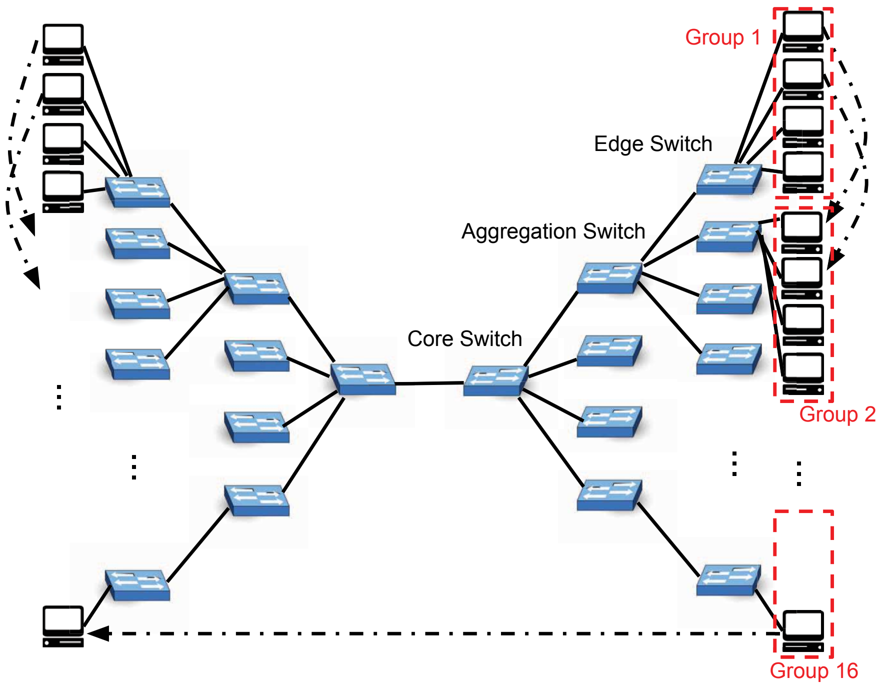
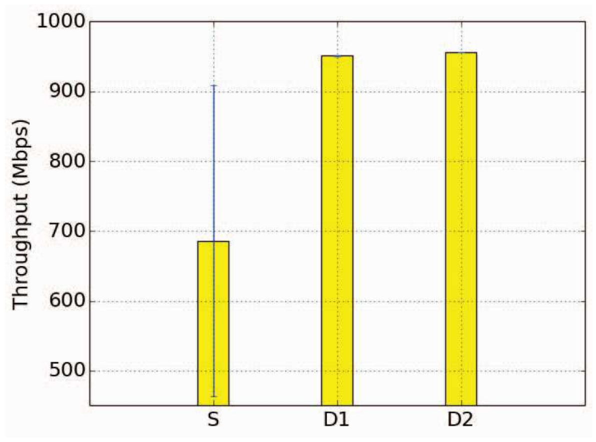


Figure 2: A double-tree network topology and traffic flows.

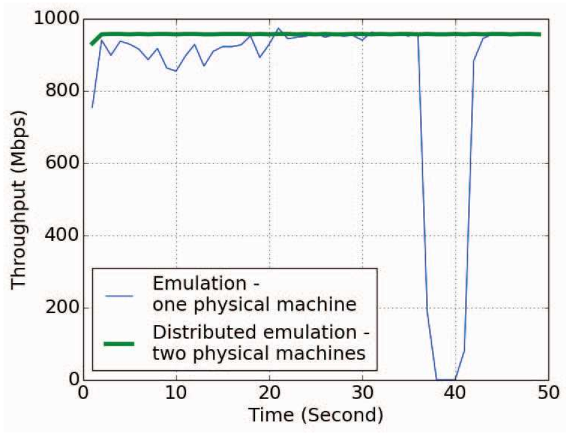
## 3.1 Distributed Emulation

We first evaluate the distributed emulation mode of our testbed with a set of throughput measurement experiments in a double-tree network and compare the results of the same scenario emulated on a single physical machine. The double-tree network contains two tree topologies whose root nodes are connected (see Figure 2). We set the degree and fan out to 4 for each tree. Each tree is composed of 1 core switch (root node), 4 aggregation switches, 16 (i.e., $4^2$) edge switches, and 64 (i.e., $4^3$) hosts. We set the link bandwidth as follows: 1 Gbps between the hosts and the edge switches to 1 Gbps, 4 Gbps between the edge switches and the aggregation switches, 16 Gbps between the aggregation switches and the core switches, and 1 Gbps between the two core switches. We use iperf to generate traffic and measure the throughput. In each tree, we arrange the hosts into 16 groups as shown in Figure 2, and each group has four hosts. We set the traffic flow in the way that $host_i$ in $group_j$ (where $j$ is an odd number) sends TCP traffic to $host_i$ in $group_{j+1}$. The only exception is that $host_4$ in $group_{16}$ of the right-hand side tree sends traffic to $host_4$ in $group_{16}$ of the left-hand side tree. The traffic flows are also illustrated in Figure 2.

We execute the same set of experiments in two modes. In the non-distributed mode, the double-tree topology network is emulated using Mininet on a single physical machine. In the distributed mode, each tree is emulated using Mininet on a physical machine, and each root node (i.e., an OVS switch instance) is attached to a Gigabit physical network interface, and the two interfaces are connected by an Ethernet cable. Figure 3a shows the mean throughput of all the flows in both modes as well as the standard deviation. Figure 3b plots the throughput of one particular flow at each second interval. Since there is no bandwidth contention among the flows, ideally each flow's throughput is close to 1 Gbps. However, we observe that the mean throughput produced by the single-physical-machine-based emulation is 685 Mbps with a large standard deviation of 222 Mbps. In the distributed emulation mode, both trees achieve the mean throughput of over 900 Mbps (951 Mbps for the left sub-tree and 955 Mbps for the right sub-tree) with a very little variation as shown in Figure 3a. In Figure 3b, we can see that the flow throughput in the single-physical-machine-based emulation has a sharp drop between 38 to 40 second. The drop is due to the CPU contention among the hosts when generating numerous close-to-line-rate traffic flows. With



(a) Mean flow throughput in the double-tree topology network scenario.



(b) A sample trace of one flow throughput over time in the distributed emulation mode.

Figure 3: Measured flow throughput. S = Emulation on a single physical machine; D = Distributed Emulation (D1: left-side servers and right-side clients, D2: left-side clients and right-side servers).

more available physical resources provided by the two machines, the experimental results in the distributed emulation mode have better fidelity, i.e., the throughput remains stable and is always close to the line rate.

## 3.2 Hardware-in-the-loop Emulation

We evaluate the hardware-in-the-loop emulation feature of our testbed with a set of throughput measurement experiments in a switch-chain-topology-based network and compare the results of the same scenario with the emulation-only testbed settings. The switch chain topology contains $N$ Open vSwitches (OVS) and two hosts at both ends as the traffic source and sinks (see Figure 4). All the link bandwidth is set 1 Gbps. For the emulation-only case, we emulate the entire network in Mininet on the server machine, including container-based virtual hosts and OVS instances. For the hardware-in-the-loop emulation case, we emulate the two hosts as two virtual hosts in Mininet on the server machine and construct all the switches using the OVS instances in a Pica 8 switch hardware. Each virtual host is attached to a Gigabit physical network interface, which is connected to a Pica 8 switch port using a network cable.

In this set of experiments, we vary the number of switches $N$, and use iperf to measure the TCP throughput between the two hosts. We plot the results for $N = 2$ and $N = 24$ in Figure 5. Figure 5a shows the mean throughput with standard deviation. The mean throughputs are 931 Mbps when $N = 2$ and 927 Mbps when $N = 24$ for the emulation-only case, and 941 Mbps when $N = 2$ and 940 Mbps when $N = 24$ for the hardware-in-the-loop emulation case. In both cases, the mean throughputs are close to the line rate, and the hardware-in-the-loop emulation is slightly better. However, when $N = 24$, we observe the standard derivation in the emulation-only case is 15 Mbps, which is larger than the hardware-in-the-loop emulation case with the value of 4.3 Mbps. In fact, we observe the same behavior in all the experiments with different values of $N$. Figure 5b shows a sample trace of the throughput at each second interval for the $N = 24$ case. We can see that the hardware-in-the-loop emulation results are more stable, i.e., closer to the expected behavior on a physical testbed, while the emulation-only settings introduce unrealistic variance to the flow throughput. It is because when the number of active virtual switches exceeds the number of available parallel processors in the physical machine, the system will experience delayed event execution since CPU resources are multiplexed in time by the OS scheduler.

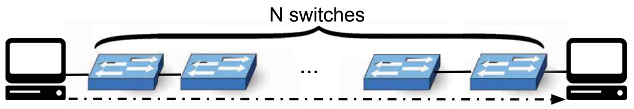
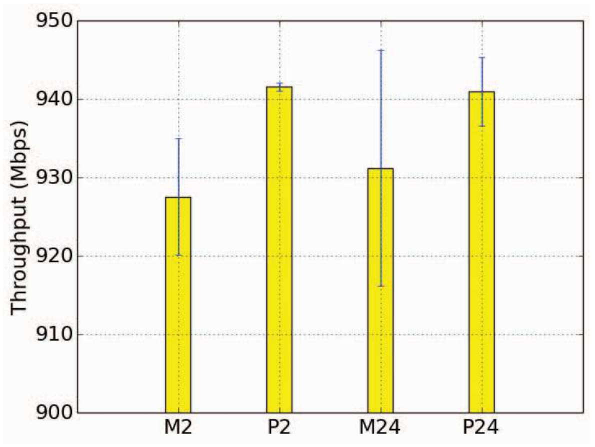
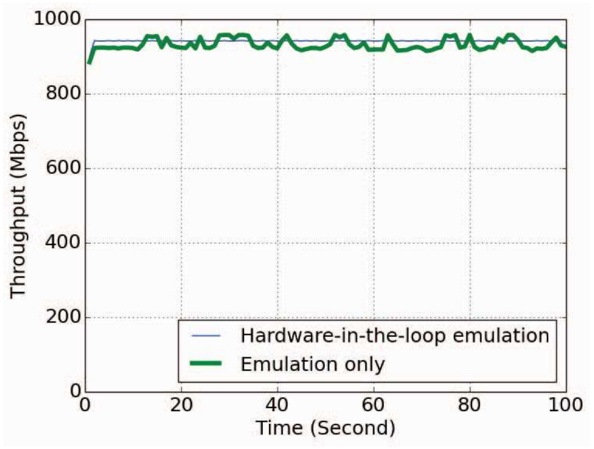


Figure 4: An $N$-switch chain network topology and traffic flows.



(a) Mean flow throughput in the switch chain topology network scenario.



(b) A sample trace of one flow throughput over time.

Figure 5: Measured flow throughput in the switch chain topology network. Type of the switch: M = emulated virtual switch, P = physical switch. Number of switches: $N = \{2, 24\}$.

## 3.3 Hardware Switch Placement

The hardware-in-the-loop feature enhances the fidelity of our emulation testbed. In the set of experiments, we further investigate how the different hardware placements (i.e., replacing virtual switches with Pica 8 hardware switches) in the same target network can affect the experimental results. We create a star topology network consisting of 4 hosts and 5 switches (see Figure 6). All the link bandwidth is set to 1 Gbps. Every host is configured to be a server and a client and generates and forwards packets to their neighboring hosts as shown in Figure 6.

We run experiments in the aforementioned network scenario with three different hardware placement strategies: (1) all the switches replaced by physical switches (baseline case), (2) the core switch replaced by a physical device, and (3) one edge switch replaced by a physical device. We compare the mean flow throughput and plot the results in Figure 7. The first strategy produces the baseline behavior, which is nearly 950 Mbps mean throughput with little variation. The second strategy produces a smaller mean throughput of 905 Mbps with a larger standard deviation of 71 Mbps. The results in the third strategy degrade even more, with an average throughput of 895 Mbps and a standard deviation of 103 Mbps. Since the core switch handles four flows and the edge switch only handles two flows, replacing the core switch with a physical device leads to better fidelity. We observe that the different hardware placement strategies in the same network scenario may result in different performance, and replacing the heavily-loaded emulated network nodes with capable hardware devices is a preferred strategy.

## 4 CASE STUDY: REPRODUCING HEDERA SCHEDULING SCHEME FOR DATA CENTER NETWORKS

In this section, we take the same approach in Mininet case study (Handigol et al. 2012) to demonstrate how to use our testbed to reproduce existing network research. We present the results of Hedera, a scalable and

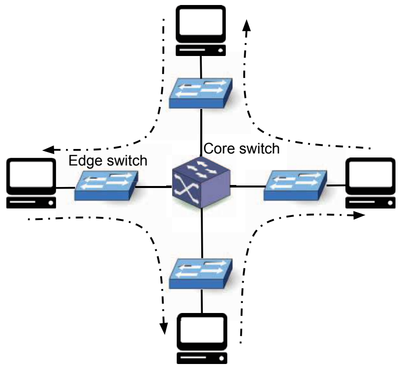


Figure 6: A star network topology and traffic flows.

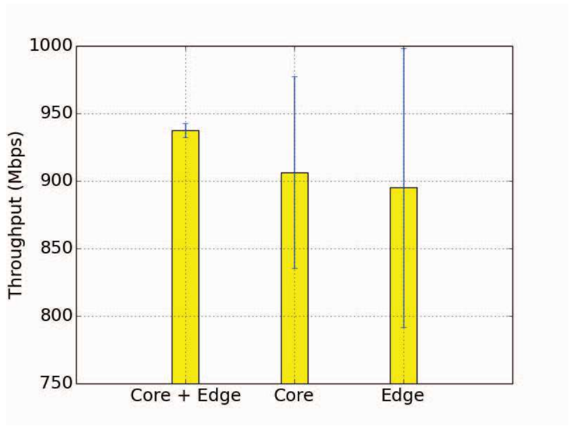


Figure 7: Mean flow throughput in the star topology network scenario. Core/Edge means replacing the emulated core/edge switch with a hardware switch.

adaptive network traffic flow scheduling system to achieve bandwidth efficiency for data center networks (Al-Fares et al. 2010). Equal Cost Multipath (ECMP) (Hopps 2000) is the commonly-used scheduling scheme in data center network. With ECMP, the path that a flow to take is selected by the hash of a packet header. Hedera improves the efficiency of network resource allocation with a tight control-loop, including (1) detection of large flows, (2) estimation of bandwidth demands of those flows, and (3) calculation of desired paths and network update. To show the advantages of Hedera over ECMP, the authors in the original paper constructed fat-tree (Vahdat, Al-Fares, and Loukissas 2013) networks using both a physical testbed and a simulation testbed, and compared the aggregate bisection bandwidth. In this paper, we implement both Hedera and ECMP scheduling schemes as applications on the ONOS SDN controller. To reproduce the experiments in the original paper, we set up the same network scenario in our testbed using both emulation-only mode and hardware-in-the-loop mode (i.e., replacing the core switches with Pica 8 switches). The network topology is a 16-host, 3-stage fat-tree as shown in Figure 8 (Vahdat, Al-Fares, and Loukissas 2013). We set all the links to 1 Mbps, and randomly generate 16 TCP flows using iperf for 60 seconds. The average bisection bandwidth is measured for the middle 40 seconds.

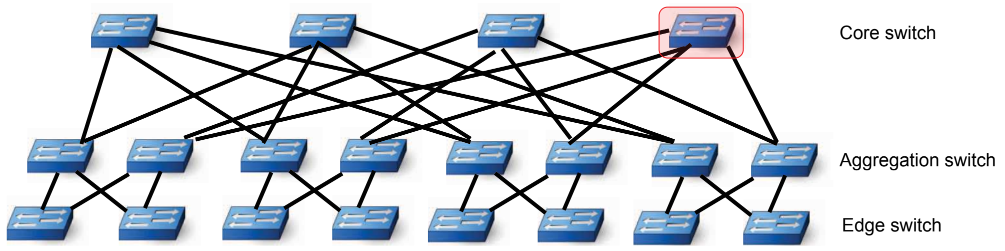
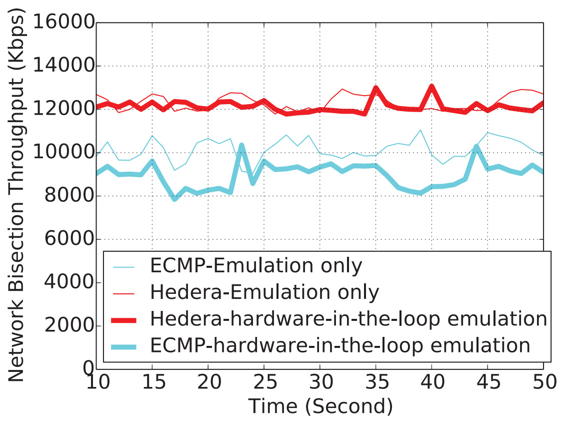
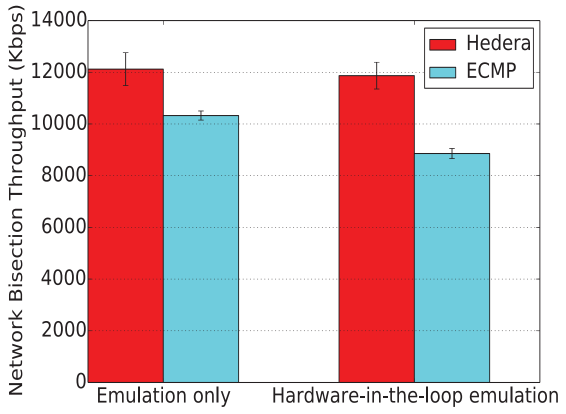


Figure 8: A fat-tree network topology for Hedera experiments.

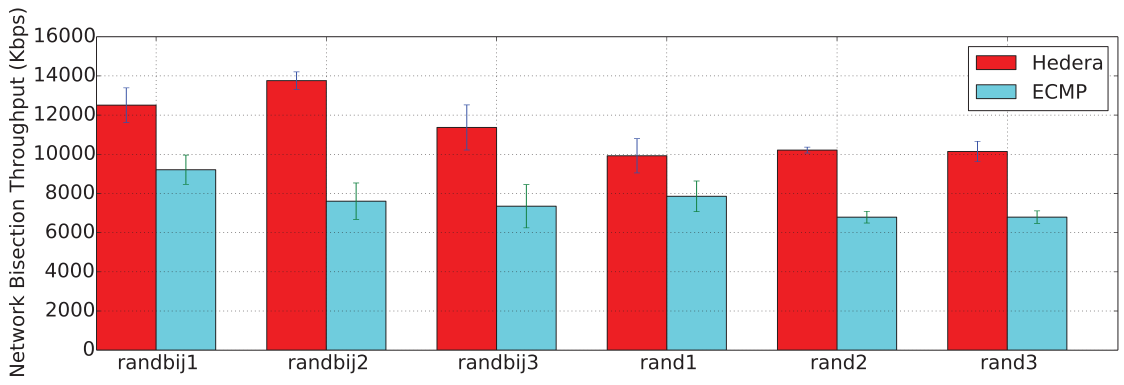
Figure 9a presents the measured bisection bandwidth over time for both Hedera and ECMP schemes using two emulation modes. We observe that in both modes Hedera always generates higher bisection throughput (around 12 Mbps) than ECMP (around 10 Mbps) during the entire data transmission. Figure 9b presents the average aggregate throughput of ten tests and the standard derivation for both Hedera and ECMP using two emulation modes. We observe the same behavior as the original paper (Al-Fares, Radhakrishnan, Raghavan, Huang, and Vahdat 2010) that the set of paths selected by Hedera manage to utilize the network bandwidth more efficiently. In addition, we also evaluate the performance of Hedera and



(a) Network bisection bandwidth over time.



(b) Mean network bisection bandwidth.



(c) Mean network bisection bandwidth measurement using hardware-in-the-loop emulation. randbij = random bijective traffic pattern, rand = random selection of destination with uniform probability

Figure 9: Reproducing Hedera scheduling scheme in a 16-host fat-tree network.

ECMP for a variety of randomized communication patterns using the hardware-in-the-loop emulation mode, and plot the experimental results in Figure 9c. `randbij` stands for the random bijective traffic pattern, and `rand` stands for the random selection of destination host with uniform probability. The throughputs of the `randbij` pattern are higher than the `rand` patterns because flows may have the same destination in the `rand` patterns. We reproduce the same behaviors in the original paper (Al-Fares, Radhakrishnan, Raghavan, Huang, and Vahdat 2010) in our testbed that Hedera has achieved higher throughput than ECMP in all the experiments with both communication traffic patterns.

## 5 CONCLUSION AND FUTURE WORK

We develop a hybrid network testbed combining lightweight container-based network emulation and physical devices for SDN researchers to conduct high fidelity and reproducible networking experiments. Our future work includes integrating the testbed with cyber-physical system simulators, such as electric power grid and traffic simulators, to facilitate our research work on the cyber-security evaluation of those systems. We will also explore means to enable the testbed to support traditional network experiments (e.g., software routers and distributed protocols/applications) in addition to SDN-based application systems. In addition, we will evaluate the scalability of the testbed in large-scale network settings.

## ACKNOWLEDGMENTS

This work is partly sponsored by the Air Force Office of Scientific Research (AFOSR) under Grant YIP FA9550-17-1-0240, the National Science Foundation (NSF) under Grant CNS-1730488, and a cooperative agreement between IIT and National Security Research Institute (NSRI) of Korea. 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 AFOSR, NSF, and NSRI.

## REFERENCES

Al-Fares, M., S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. 2010. "Hedera: Dynamic Flow Scheduling for Data Center Networks". In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, 19–19. Berkeley, CA, USA: USENIX Association.

Berde, P., M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow et al. 2014. "ONOS: Towards an Open, Distributed SDN OS". In *Proceedings of the third Workshop on Hot Topics in Software Defined Networking*, 1–6. ACM.

Berman, M., J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar. 2014. "GENI: A Federated Testbed for Innovative Network Experiments". *Computer Networks* 61:5–23.

Carrozzo, G., R. Monno, B. Belter, R. Krzywania, K. Pentikousis, M. Broadbent, T. Kudoh, A. Takefusa, A. Vieo-Oton, C. Fernandez, B. Puvpe, and J. Tanaka. 2014. "Large-scale SDN experiments in federated environments". In *Proceedings of the 2014 International Conference on Smart Communications in Network Technologies (SaCoNeT)*, 1–6. IEEE.

Chan, M. C., C. Chen, J. X. Huang, T. Kuo, L. H. Yen, and C. C. Tseng. 2014. "OpenNet: A Simulator for Software-Defined Wireless Local Area Network". In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, 3332–3336. IEEE.

DOE Accessed 2017. "ESnet: Energy Sciences Network". http://www.es.net/network-r-and-d/experimental-network-testbeds/100g-sdn-testbed.

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

Hopps, Christian E 2000. "Analysis of an Equal-cost Multi-path Algorithm". https://tools.ietf.org/html/rfc2992.

Jin, D., and D. M. Nicol. 2015. "Parallel Simulation and Virtual-Machine-Based Emulation of Software-Defined Networks". *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 26 (1): 8:1—-8:27.

Piotr Jurkiewicz 2013. "Link Modeling using NS-3". https://github.com/mininet/mininet/wiki/Link-modeling-using-ns-3.

Mikovic, J., P. G. Kannan, C. Mun Choon, and K. Sklower. 2017. "Enabling SDN Experimentation in Network Testbeds". In *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, 7–12. ACM.

OMNet++ Accessed 2013. "OMNet++ SDN Projects". http://omnet-manual.com/omnet-sdn-projects/.

Ricci, R., and E. Eide. 2014. "Introducing CloudLab: Scientific Infrastructure for Advancing Cloud Architecturesand Applications". *;login:* 39 (6): 36–38.

Joel Sommers 2013. "FS: a Network Flow Record Generator". https://github.com/jsommers/fs.

Suñé, M., L. Bergesio, H. Woesner, T. Rothe, A. Köpsel, D. Colle, B. Puype, D. Simeonidou, R. Nejabati, M. Channegowda, M. Kind, T. Dietz, A. Autenrieth, V. Kotronis, E. Salvadori, S. Salsano, M. Körner, and S. Sharma. 2014. "Design and Implementation of the OFELIA FP7 Facility: The European OpenFlow testbed". *Computer Networks* 61:132–150.

Vahdat, Amin and Al-Fares, Mohammad and Loukissas, Alexander 2013, July 9. "Scalable Commodity Data Center Network Architecture". US Patent 8,483,096.

Wang, S.-Y., C.-L. Chou, and C.-M. Yang. 2013. "EstiNet Openflow Network Simulator and Emulator". *IEEE Communications Magazine* 51 (9): 110–117.

Yan, J., and D. Jin. 2015. "VT-Mininet: Virtual-Time-Enabled Mininet for Scalable and Accurate Software-Define Network Emulation Categories and Subject Descriptors". In *Proceedings of the ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR)*, 27:1—-27:7.

Yang, Q. 2017, May. "A Hardware-in-the-Loop Software-Defined Networking Testing and Evaluation Framework". Master's thesis, Illinois Institute of Technology.

## AUTHOR BIOGRAPHIES

**XIAOLIANG WU** is a research assistant in the Department of Computer Science at the Illinois Institute of Technology. His research focuses on network emulation and software-defined network. He received an M.S. degree in Computer Science from the Illinois Institute of Technology in 2017. His e-mail address is xwu64@hawk.iit.edu.

**QI YANG** is a master student and a graduate research assistant of the Department of Computer Science at the Illinois Institute of Technology. His interests include cyber security and software-defined networks. His email address is qyang18@hawk.iit.edu.

**XIN LIU** is a Ph.D. candidate in the Department of Computer Science at the Illinois Institute of Technology. His interests lie in simulation and emulation of software-defined networks. His email address is xliu125@hawk.iit.edu.

**DONG (KEVIN) JIN** is an Assistant Professor in the Computer Science Department at the Illinois Institute of Technology. He obtained his Ph.D. in Electrical and Computer Engineering from the University of Illinois at Urbana-Champaign. His research interests include simulation modeling and analysis, trustworthy cyber-physical critical infrastructures, software-defined networking, and cyber-security. He is a recipient of the Air Force Office of Scientific Research (AFOSR) Young Investigator Program (YIP) award. His email address is dong.jin@iit.edu.

**CHEOL WON LEE** is a Principal Member of Engineering Staff in National Security Research Institute of Korea. He holds a Ph.D. degree in Computer Engineering with Cyber Security specialization in Ajou University, Korea. His research interests lie in the areas of smart grid security, critical infrastructure protection, and cyber-physical security. His email address is cheolee@nsr.re.kr.