EstiNet OpenFlow Network Simulator and Emulator

Shie-Yuan Wang, National Chiao Tung University

Chih-Liang Chou and Chun-Ming Yang, EstiNet Technologies, Inc.

ABSTRACT

In this article, we introduce the EstiNet OpenFlow network simulator and emulator, and present its support for testing the functions and evaluating the performances of software-defined networks' OpenFlow controller's application programs. EstiNet uses an unique kernel re-entering simulation methodology to enable unmodified real applications to run on nodes in its simulated network. As a result, without any modification, real NOX/POX or Floodlight OpenFlow controllers can readily run on a host in an EstiNet simulated network to control thousands of simulated OpenFlow switches. EstiNet has the characteristics of a simulator and an emulator at the same time. It combines the advantages of the simulation and emulation approaches without their respective shortcomings. EstiNet uses real OpenFlow controller programs, real network application programs, and the real TCP/IP protocol stack in the Linux kernel to generate correct, accurate, and repeatable SDN application performance results. In this article, we compare EstiNet with ns-3 and Mininet regarding their capabilities, performance, and scalability.

INTRODUCTION

Software-defined networks (SDNs) [1] are a new type of network that can be programmed by a software controller according to various needs and purposes. The goal of SDNs is to facilitate innovations in network architecture and protocol designs. To achieve this goal, the OpenFlow protocol [2] has been proposed to define the internal architecture of an OpenFlow switch, and the messages exchanged between an OpenFlow controller and OpenFlow switches. In an OpenFlow network, because the operation and intelligence of the network are fully controlled by an OpenFlow controller, the correctness and efficiency of the functions implemented by the controller must be fully tested before its use in a production network.

Testing the correctness and evaluating the performance of a network protocol can be performed through several approaches. One approach is performing these tests over an experimental testbed (e.g., Emulab [3] and PlanetLab [4]). Although this approach uses real devices running real operating systems and applications, and can generate more realistic testing results, the cost of building a large experimental testbed is huge, and generally the testbed is not easily accessible to many users.

Another common approach is via simulation, in which the operations of real devices and their interactions are modeled and executed in a software program. The simulation approach has many advantages such as low cost, and being flexible, controllable, scalable, repeatable, accessible to many users, and faster than real time in many cases. However, if the modeling of real devices is not accurate enough, the simulation results may differ from the experimental results. To overcome this problem, the emulation approach may be used. Emulation differs from simulation in that an emulation is like an experiment and thus must be executed in real time, while simulation speed can be faster or slower than real time. Furthermore, in an emulation some real devices running real operation systems and application programs will interact with some simulated devices. In contrast, in a simulation generally no real operation systems or applications are involved.

In this article, we introduce the EstiNet OpenFlow network simulator and emulator [5]. EstiNet uses an unique approach to testing the functions and performances of OpenFlow controllers. By using an innovative simulation methodology, which is called *kernel re-entering* [6], EstiNet combines the advantages of both the simulation and emulation approaches. In a network simulated by EstiNet, each simulated host can run the real Linux operating system, and any UNIX-based real application program can readily run on a simulated host without any modification. With these unique capabilities, EstiNet's simulation results are as accurate as those obtained from an emulation while still preserving the many advantages of the simulation approach. For example, in EstiNet the simulation need not be executed in real time, but instead can be faster or slower than real time. This capability makes it able to correctly simulate the performance of an OpenFlow network that consists of a very large number of OpenFlow switches and hosts.

In testing OpenFlow controllers, since the

0163-6804/13/$25.00 © 2013 IEEE

widely used NOX/POX [7] and Floodlight [8] OpenFlow controllers are real application programs runnable on Linux, they can readily run on a host in an EstiNet simulated network to control thousands of simulated OpenFlow switches. In EstiNet, because these real OpenFlow controllers are tested and evaluated in simulations rather than in emulations, they can correctly control thousands of simulated OpenFlow switches based on the simulation clock, even though the simulation clock has to advance at a speed slower than real time due to insufficient computation resource on a machine. The performance results of a simulated OpenFlow network managed by these OpenFlow controllers are correct, accurate, and repeatable in EstiNet. These performance results can be correctly explained based on the parameter settings (link bandwidth, delay, downtime, etc.) and configurations (e.g., network size, mobility pattern, or speed) of the simulated OpenFlow network.

We have used EstiNet to successfully perform functional validation and performance evaluation of several protocols that are implemented by NOX/POX as components. For example, we have studied the Learning Bridge Protocol (LBP) and STP (Spanning Tree Protocol) of NOX/POX, and identified several of their design and implementation flaws. (Note: In the SDN community, to be precise, these components are called *controller applications*. However, to save space in the article, when there is no ambiguity, we just use *controller* to refer to a controller running with some components.) In this article, we compare EstiNet with other similar tools (ns-3 [9] and Mininet [10]) regarding their capabilities. Then we run the network topology discovery component of NOX/POX over an $N \times N$ grid network, where $N = 5, 6, 7, ..., 15$, to study and compare the execution performance and scalability of EstiNet and Mininet. As shown in the rest of this article, when one studies the performance of an SDN OpenFlow controller's application, EstiNet generates correct, accurate, and repeatable performance results, but Mininet cannot. In addition, EstiNet is much more scalable than Mininet when studying large OpenFlow networks.

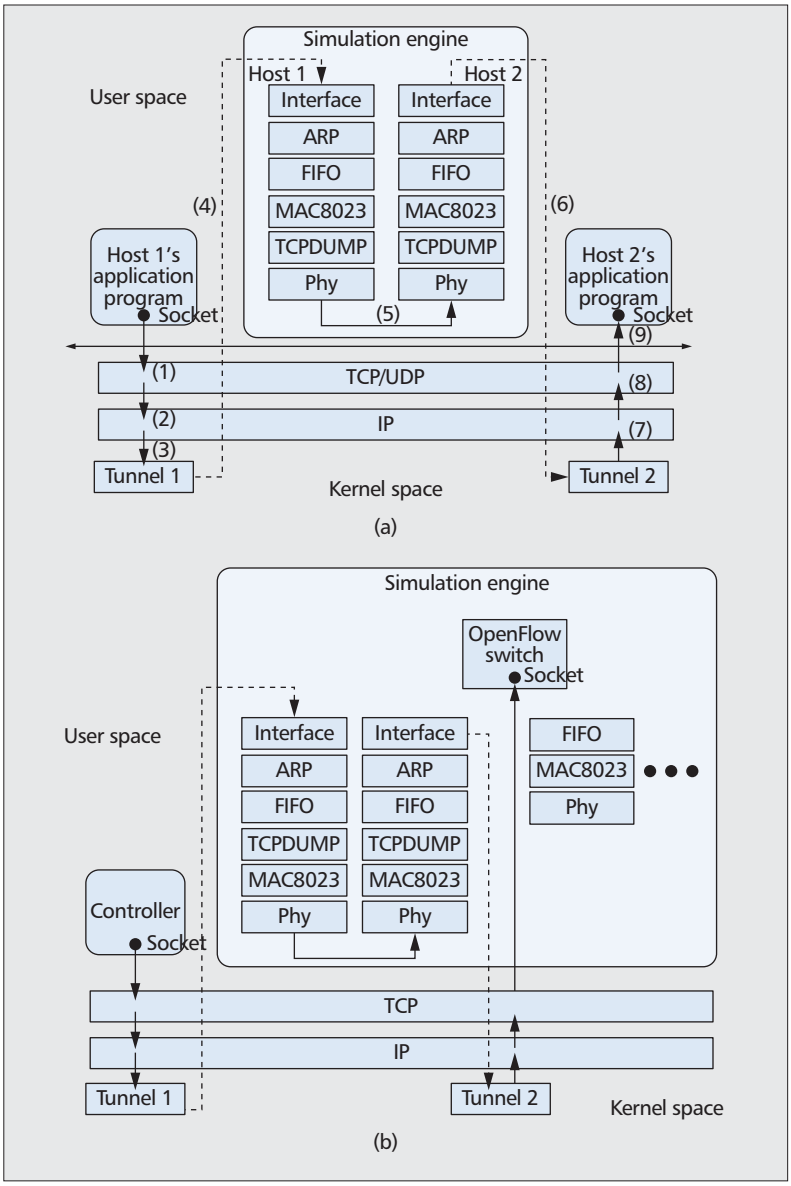

Figure 1. *Simulation architecture of EstiNet: a) the host-to-host case; b) the controller-to-OpenFlowSwitch case.*

Simulation Architecture of EstiNet

To implement the kernel re-entering methodology [6], EstiNet uses tunnel network interfaces to automatically intercept the packets exchanged by two real applications and redirect them into the EstiNet simulation engine. As shown in Fig. 1a, inside the EstiNet simulation engine, a protocol stack composed of the medium access control/physical (MAC/PHY) layers along with other layers below the IP layer is created for each simulated host. Packets to be sent out on host 1 are sent out to the output queue of tunnel interface 1, from which the simulation engine will fetch them later. After fetching a packet from tunnel interface 1, the simulation engine processes the packet through the protocol stack created for host 1 to simulate the MAC/PHY and many other mechanisms of the network interface used by host 1. For example, the effects of the link delay, link bandwidth, link downtime, and link bit error rate (BER) are all simulated in the PHY module. The PHY module of host 1 will deliver the packet to the PHY module of host 2 after the link delay plus the transmission time of the packet on this link based on the simulation clock. Then the packet will be processed from the PHY module up to the interface module, where it is written back into the kernel via tunnel interface 2. The packet will then go through the IP/TCP/socket layers and finally be received by the application running on host 2, which ends its journey. By this methodology, all Linux-based real applications can run on a simulated network in EstiNet without any modification, and they all use the real TCP/IP protocol stack in the Linux kernel to create their TCP connections.

Figure 1b shows how we extend this methodology to support running a real OpenFlow controller on an EstiNet simulated network. Since

All messages exchanged between a real OpenFlow controller and a simulated OpenFlow switch are accurately scheduled based on the simulation clock. Therefore, the results of functional validation and performance evaluation of a real OpenFlow controller are correct and repeatable over EstiNet.

real OpenFlow controllers such as NOX/POX and Floodlight are normal application programs, they readily run on a simulated host in EstiNet without any modification. However, because a real OpenFlow switch needs to set up a TCP connection to the OpenFlow controller to receive its messages, we simulate the operations of each OpenFlow switch inside the simulation engine and let it create an OpenFlow TCP socket bound to a network interface (in this example, the used network interface is tunnel interface 2). With this design, in EstiNet a simulated OpenFlow switch can set up a real TCP connection to a real OpenFlow controller to receive its messages. All messages exchanged between a real OpenFlow controller and a simulated OpenFlow switch are accurately scheduled based on the simulation clock. Therefore, the results of functional validation and performance evaluation of a real OpenFlow controller are correct and repeatable over EstiNet.

Capability Comparison with Related Tools

Currently, very few network simulators support the OpenFlow protocol; the most notable one is ns-3 [9]. The ns-3 tool is the most widely used network simulator in the world. There is a project of ns-3 for supporting the OpenFlow protocol, and the version of the OpenFlow protocol supported is 0.89, which is quite old as the latest version of OpenFlow as of this writing is already 1.3.1. Ns-3 simulates the operations of an OpenFlow switch by compiling and linking an OpenFlow switch C++ module with its simulation engine code. To simulate a real OpenFlow controller, ns-3 also implements it as a C++ module, and compiles and links it with its simulation engine code. In fact, all devices/objects simulated in ns-3 are implemented as C++ modules compiled and linked together with its simulation engine code to form a user-level executable program (i.e., ns-3).

Because the ns-3 program is a user-level program, and a real OpenFlow controller such as NOX/POX is also a user-level program, a real OpenFlow controller program cannot be compiled and linked together with the ns-3 program to form a single executable program. As a result, a real OpenFlow controller cannot readily run without modification on a node in a network simulated by ns-3. This is why ns-3 has to implement its own OpenFlow controller from scratch as a C++ module. This approach wastes much time and effort to re-implement widely used real OpenFlow controllers. In addition, the running behavior of a re-implemented OpenFlow controller module in ns-3 may not be the same as the behavior of a real OpenFlow controller because the former is a much-simplified abstraction of the latter. For example, as documented in ns-3, STP and the multiprotocol label switching (MPLS) function are not supported. As another example, in ns-3 there is no TCP connection between a simulated OpenFlow switch and its simulated OpenFlow controller. Since this is not the usage in the real world, the simulation results will differ from the real results when the TCP connection in the real world experiences packet losses or congestion.

As of this writing (April 25, 2013), from the ns-3 official web site and its OpenFlow developer forum, we see that its developers all agreed that it was too difficult to upgrade ns-3 to support OpenFlow 1.0, and so far ns-3 still cannot work with a real external OpenFlow controller. Due to these reasons, the ns-3 OpenFlow project has been suspended without progress for a long time, and it is not used by the SDN research community.

Regarding network emulators that support the OpenFlow protocol, currently there are very few such tools; the most notable one is Mininet [10]. Mininet uses the virtualization approach to create emulated hosts and uses the Open vSwitch [11] to create software OpenFlow switches on a physical server. The links connecting a software OpenFlow switch to emulated hosts or other software OpenFlow switches are implemented by using the virtual Ethernet pair mechanism provided by the Linux kernel. Because an emulated host in Mininet is like a virtual machine, real applications can readily run on it to exchange information. A real OpenFlow controller, which is also a real application, can also run on an emulated host to set up TCP connections to software OpenFlow switches to control them. With this approach, emulated hosts and software OpenFlow switches can be connected together to form a desired network topology and be controlled by a real OpenFlow controller. So far, from the ONS 2013 conference [12] held in April 2013, one can see that currently Mininet is the most popular tool used by the SDN research community.

Although Mininet can be used for rapid prototyping of SDNs, it has several limitations. As stated in [10], the most significant limitation of Mininet is its lack of performance fidelity because it provides no guarantee that an emulated host in Mininet that is ready to send a packet will be scheduled promptly by the operating system to send the packet, and it provides no guarantee that all software OpenFlow switches in Mininet will forward packets at the same rate. The packet forwarding rate of a software OpenFlow switch in Mininet is unpredictable and varies in every experimental run as it depends on the CPU speed, the main memory bandwidth, the number of emulated hosts and software OpenFlow switches that must be multiplexed over a CPU in Mininet, and the current system activities and load. As a result, Mininet can only be used to study the behavior of an OpenFlow controller but cannot be used to study any time-related network/application performance. We show that Mininet is unscalable and will become unreliable when the number of software OpenFlow switches controlled by it increases.

In contrast, EstiNet combines the advantages of both the simulation and emulation approaches without their respective shortcomings. As in an emulation, in EstiNet a real OpenFlow controller can readily run without modification to control simulated OpenFlow switches, and real applications can readily run on hosts running a real operating system to generate realistic network traffic. However, the operations and inter-

	EstiNet	ns-3	Mininet
OpenFlow specification	1.1.0/1.0.0	0.8.9	1.0.0
Simulation mode	✓	✓	—
Emulation mode	✓	—	✓
Compatible with real-world controllers	✓	—	✓
Result repeatable	✓	✓	—
Scalability	**High** by single process	**High** by single process	**Middle** by multiple processes
Performance result correctness	✓	**No Spanning Tree Protocol and no real-world controller**	**No performance fidelity**
GUI support	✓ • Configuration • Observation	✓ • Observation only • Using C++ for configuration	✓ • Observation only • Using Python for configuration

Table 1. *A comparison of EstiNet, ns-3, and Mininet.*

EstiNet uses the kernel reentering methodology to use a single kernel to support multiple hosts and its simulation engine process can simulate multiple OpenFlow switches. As a result, it is highly scalable.

actions among these real applications, the real OpenFlow controller, the OpenFlow switches, and the hosts and links in a studied network are all scheduled by the EstiNet simulation engine based on its simulation clock, rather than multiplexed and executed in an unpredictable way by the operating system. For this reason, differing from Mininet, EstiNet generates time-related OpenFlow performance results correctly, and the results are repeatable.

In Table 1, we compare the capabilities of EstiNet, ns-3, and Mininet according to their latest developments. Most comparison results are self-explanatory; thus, we only explain the scalability and GUI comparison results. EstiNet uses the kernel re-entering methodology to use a single kernel to support multiple hosts, and its simulation engine process can simulate multiple OpenFlow switches. As a result, it is highly scalable. Ns-3 is also highly scalable as its simulated hosts, OpenFlow switches, and controller are all implemented as C++ modules and linked together as a single process. In contrast, Mininet needs to run up a shell process (e.g., /bin/bash) to emulate each host and needs to run two user-space Open vSwitch processes (one for the data path and the other for the control path) to emulate each OpenFlow switch. As a result, it is less scalable than EstiNet and ns-3. (Note: Open vSwitch can also be run in the kernel mode. In such a mode, only one process needs to be run for the control path. However, according to our measurements, Open vSwitch running in the kernel mode makes Mininet less scalable and more unreliable.) We show the scalability comparison results between EstiNet and Mininet.

Regarding graphical user interface (GUI) support, which is very important for the user, EstiNet's GUI can be used to easily set up and configure a simulation case, and to observe the packet playback of a simulation run. The GUI of ns-3, on the other hand, can only be used for observation of the results, and the user needs to write C++ or scripts to set up and configure the simulation case. For Mininet, its GUI can be used for observation purposes only, and the user needs to write Python scripts to set up and configure the simulation case. One unique and useful feature of EstiNet GUI is that it can show the playback of OpenFlow control packets after a simulation is finished. By this capability, an OpenFlow controller developer can more easily test and debug her OpenFlow controller program by visualizing their behavior over EstiNet. Figure 2 shows an OpenFlow network topology that is constructed by the GUI of EstiNet.

In this example network, nodes 3, 4, 5, and 11 are simulated hosts running the real Linux operating system where real applications can run without modification. Nodes 6, 7, 8, 9, and 10 are simulated OpenFlow switches supporting the OpenFlow protocol. Node 1 is the host where NOX/POX will be running during simulation. (In the following, we call it the *controller node* for brevity.) Node 2 is a simulated legacy (normal) switch that connects all simulated OpenFlow switches together with the controller node. It forms a management network over which the TCP connection between each simulated OpenFlow switch and the controller node will be set up. All OpenFlow messages between NOX/POX and simulated OpenFlow switches are exchanged over this management network. In contrast, the network formed by simulated OpenFlow switches, simulated hosts, and the links connecting them together is the data network over which real applications running on simulated hosts will exchange their information. This figure is one snapshot of the packet animation playback of EstiNet. From this figure, one can easily see that

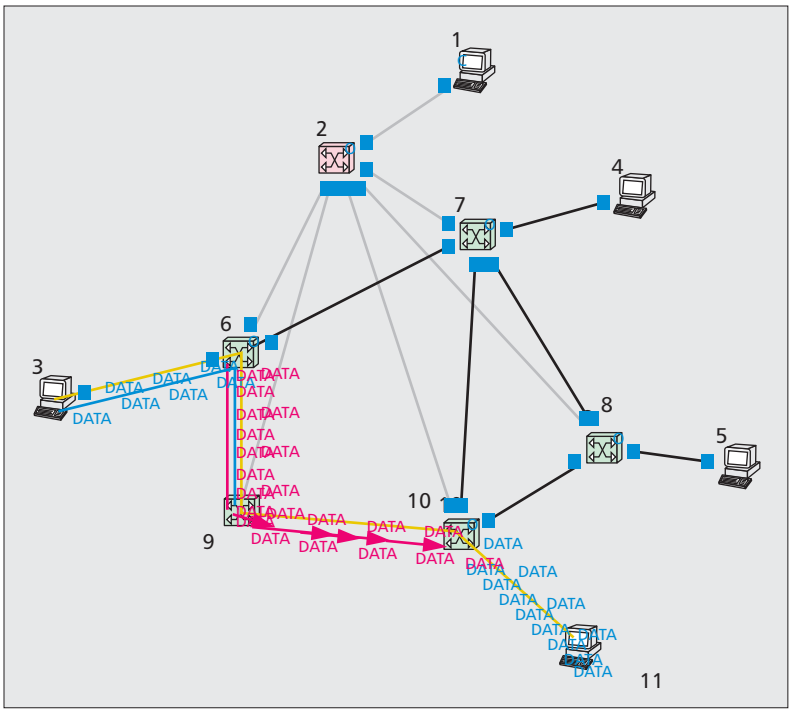

Figure 2. *An OpenFlow network topology constructed by the GUI of EstiNet.*

packets are flowing along the (host 3, switch 6, switch 9, switch 10, host 11) routing path.

As of this writing, EstiNet is run on Linux Fedora 17 with Linux kernel v. 2.6.35. The TCP version used in this kernel version is TCP cubic, and the IP protocol version used in the simulation study is IPv4. When we compare the execution performance and scalability of EstiNet and Mininet, the notebook machine used is equipped with an Intel Core i5 CPU at 2.67 GHz and 4 Gbytes of main memory.

NOX/POX's Network Topology Discovery Component

NOX/POX's Network Topology Discovery component uses the Link Layer Discovery Protocol (LLDP) [13] packets to discover the topology of an OpenFlow network. For each switch, after it is powered on and has established a TCP connection to NOX/POX, NOX/POX immediately sends it a FlowModify message to add an entry into its flow table. This flow entry will match future received LLDP packets, and its associated action is "Send the received LLDP packet to the controller." For each port of a switch, every 5 s (the LLDP transmission interval) NOX/POX sends a PacketOut message to the switch asking it to send the LLDP packet carried in the PacketOut message out of the specified port. Since every switch has already had a flow entry matching received LLDP packets, when a switch receives an LLDP packet from one of its neighboring switches, it will send the received LLDP packet to NOX/POX using the PacketIn message. With these received LLDP packets from all switches, NOX/POX builds the complete network topology and can compute a spanning tree over it if the network topology contains loops.

From the above explanations, one sees that the PacketIn and PacketOut messages triggered by the exchanges of LLDP packets on a large network can cause a heavy processing burden for NOX/POX. If the total port count of the network is N, then NOX/POX will have to send N PacketOut messages and receive N PacketIn messages every 5 s just for LLDP packets alone. The transmission interval of LLDP packets on NOX/POX is thus $5/N$ s. It is clear that as N increases, the loads imposed on NOX/POX will become heavier and heavier.

Execution Performance and Scalability Comparison of EstiNet and Mininet

To compare the execution performance and scalability of EstiNet and Mininet, we run NOX/POX's network topology discovery component on an $N \times N$ grid network, where N = 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, and 15. For a grid network of size N, it has $N * N$ OpenFlow switches. A source host is attached to the top left corner switch of the grid, and a destination host is attached to the top right corner switch of the grid. For Estinet, we set the bandwidth and delay of all links in the network to 10 Mb/s and 1 ms, respectively. For Mininet, because it has no concept of time and cannot simulate the link delay, bandwidth, downtime, and so on, these link attributes are meaningless. We run the real-life ping program on the source host to ping the destination host 100 times and record the measured ping reply delays. We call such a 100-ping process a run, and call the average of the 100 ping reply delays of a run the average ping delay (APD) of a run for easier discussion. In the following, we present the APD (milliseconds) and the main memory consumption (megabytes) of EstiNet and Mininet for different sizes of an $N \times N$ grid network.

EstiNet

EstiNet is a network simulator with its own simulation clock for controlling the execution timing of the simulated network. All real application programs (including ping) running over it are triggered by its simulation clock rather than the real-time clock. Therefore, for an $N \times N$ grid network, the 100 ping reply delays of a run are all the same (i.e., 1 ms link delay * $(N + 1)$ hops * 2 round-trips); thus, the APD of the run is the same as any measured ping reply delay of the run. When N increases, the APD of a run increases linearly with N as the ping packet needs to pass $(N + 1)$ hops to reach its destination.

As a network simulator, EstiNet accurately lets the ping program measure and report the correct ping reply delay for any ping packet over a grid network of size N. Result correctness is not a concern for EstiNet for any value of N. The only concern is how EstiNet may slow down its simulation speed and how much main memory space it may consume as N increases.

Figure 3a shows the time required to simulate 10 s of the grid network under different values

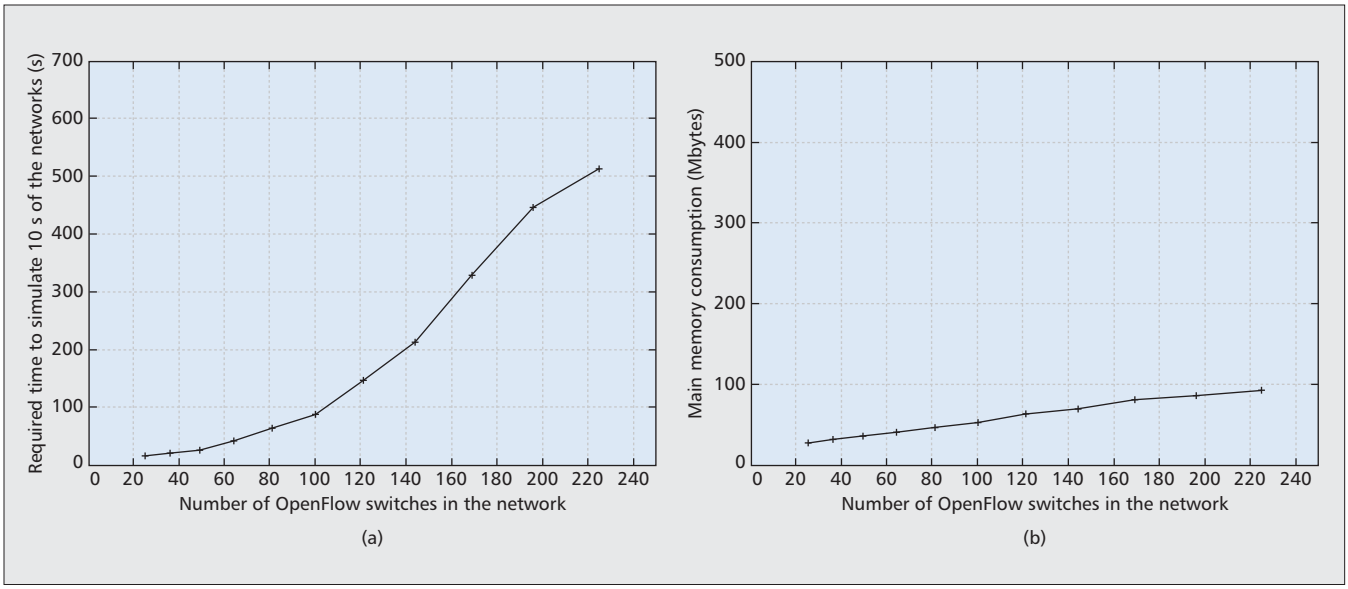

Figure 3. *Performance of EstiNet when NOX/POX controls different numbers of OpenFlow switches: a) number of OpenFlow switches in the network; b) number of OpenFlow switches in the network.*

of N, where there are $N * N$ OpenFlow switches in the grid network. As explained before, because when N increases the LLDP traffic loads imposed on NOX/POX and software OpenFlow switches increase as well, it is natural that EstiNet needs to take more time to finish simulating 10 s of the grid network when N increases.

Figure 3b shows the main memory space consumed by EstiNet to simulate the grid network under different values of N. These results are obtained using the "top" command in Linux, which can report the main memory space currently used by a process. Because each simulated OpenFlow switch needs to take a fixed amount of memory space, it is natural that EstiNet needs to take more memory space to simulate an $N \times N$ grid network when N increases. Comparing EstiNet with the memory space consumed by Mininet, shown in Fig. 4d, one sees that EstiNet consumes much less memory space than Mininet when both tools simulate/emulate the same network.

Mininet

Mininet is a network emulator without the concept of time. Using Mininet to control many software OpenFlow switches is like doing an experiment and must be performed in real time without much delay. Otherwise, the results will be different from the results obtained over a real network where NOX/POX and all OpenFlow switches run independently on different machines. Since the ping reply delay obtained over Mininet depends on many factors of the machine (CPU speed, main memory bandwidth, the number of software OpenFlow switches that need to run concurrently on the machine, and many other activities such as disk I/O accesses on the machine) and has nothing to do with the link delay and link bandwidth settings of the links in a grid network, for a grid network of size N, almost all of the 100 ping reply delays of a run measured over Mininet in real time are different from each other. (Note: ping's reply delay is reported in the unit of ms using the time=xx.yyy format.) For this reason, in addition to calculating the APD of a run, we also calculate the standard deviation of the 100 ping reply delays of a run and use StdDev to represent it.

Knowing that Mininet's performance results cannot be explained and are unrepeatable, here we do not care about the result correctness problem but instead care about at what size of N will Mininet's operations start to lag behind the real time. This is because from that size beyond, the performance results obtained over Mininet (e.g., the round-trip time, RTT, delay measured by ping) become unreliable and start to deviate from the correct value by a larger and larger difference.

As we increase the size of N to record the 100 ping reply delays of a run over Mininet, we have found that starting at $N = 7$ (i.e., when there are 49 software OpenFlow switches in the network), Mininet starts to become very unreliable. Often a run cannot finish by collecting 100 ping reply delays; instead, the ping program hangs forever without outputting any more ping reply delay. Therefore, for each grid network size of N, we performed 10 runs, counted the number of failed runs, and calculated its no-response failure rate. Figure 4b shows the failure rate of Mininet under different sizes of N. One sees that as N increases beyond 7, Mininet quickly becomes very unreliable, making it almost unusable when controlling more than 49 software OpenFlow switches.

For the 10 runs conducted for a grid network size of N, we averaged the APDs of successful runs to obtain an AAPD for this network size. Similarly, we averaged the StdDev's of successful runs to obtain AStdDev for this network size. Figure 4b shows the AAPD for different values of N. One can see that as N increases beyond 7, Mininet starts to seriously lag behind real time, and the AAPD increases very rapidly. Figure 4c shows the AStdDev for different values of N.

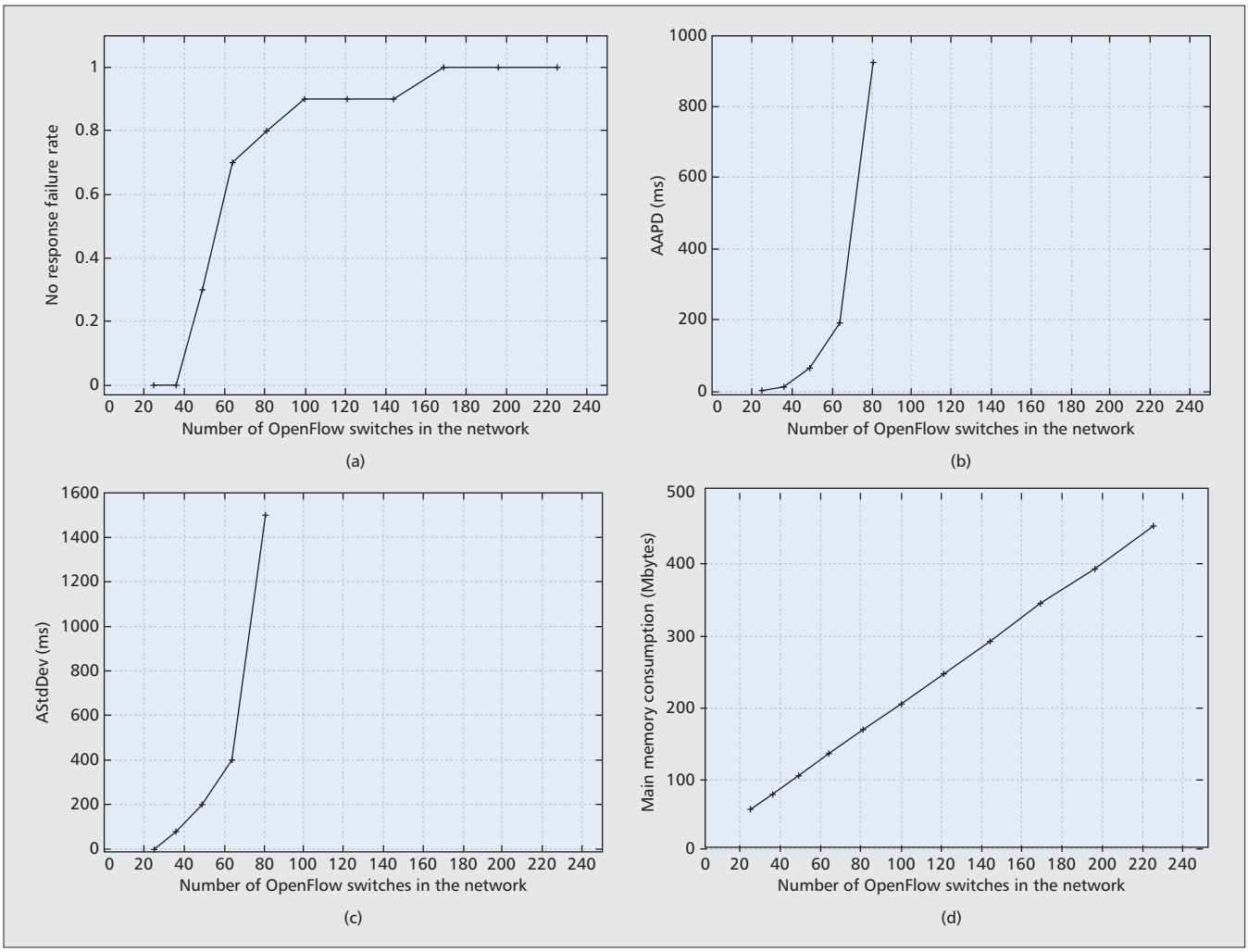

Figure 4. *Performance of Mininet when NOX/POX controls different numbers of OpenFlow switches.*

Again, one sees that as N increases beyond 7, the AStdDev increases very rapidly as well. This means that for a given run, the differences among the 100 ping reply delays become very huge, making Mininet's results very untrustworthy. Note that both Figs. 4b and 4c only show the results for the values of N up to 9. This is because when N is greater than 9, the failure rate has become so high that only one or two (or even zero) out of 10 runs are successful, which makes the calculation of AAPD and AStdDev meaningless. Figure 4d shows the memory space consumed by Mininet for different values of N. Compared with Fig. 3b, one sees that Mininet consumes much more memory space than EstiNet when they emulate/simulate the same network.

Conclusion

In this article, we present the EstiNet OpenFlow network simulator and emulator, and explain how its unique kernel re-entering methodology enables real OpenFlow controllers such as NOX/POX or Floodlight to control thousands of its simulated OpenFlow switches. By this methodology, EstiNet combines the advantages of both the simulation and emulation approaches without their respective shortcomings. EstiNet is a useful tool to test the functions and evaluate the performance of an SDN OpenFlow controller's application programs. Its generated performance results can be correctly explained based on the network parameters settings and the protocol design of the used SDN OpenFlow controller's application programs. In this article, we compare EstiNet with ns-3 and Mininet from many aspects. Specifically, we focus on the comparison of EstiNet and Mininet and show that:

- EstiNet generates correct performance results, while Mininet's performance results are untrustworthy.
- EstiNet is much more scalable than Mininet when studying large OpenFlow networks.

EstiNet can also operate in the emulation mode, in which a machine running a real OpenFlow controller such as NOX/POX or Floodlight (or even a network virtualization tool such as FlowVisor [14]) can control an OpenFlow network simulated by EstiNet (in real time) on another machine. These two machines are connected by a physical wire such as an Ethernet cable so that OpenFlow messages can be exchanged between them. Due to space limitations, the details of this kind of usage are not presented in this article.

References

[1] ONF, "Software-Defined Networking: The New Norm for Networks," white paper, Apr. 13, 2012.
[2] N. Mckeown *et al.*, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Comp. Commun. Rev.*, vol. 38, issue 2, Apr. 2008.
[3] B. White *et al.,* "An Integrated Experimental Environment for Distributed Systems and Networks," *Proc. 5th Symp. Op. Sys. Design and Implementation*, Boston, MA, Dec. 2002, pp. 255–70.
[4] B. Chun *et al.*, "PlanetLab: an Overlay Testbed for Broad-Coverage Services," *ACM SIGCOMM Comp. Commun. Rev.*, vol. 33, issue 3, July 2003.
[5] EstiNet 8.0 OpenFlow Network Simulator and Emulator, EstiNet Technologies Inc., http://www.estinet.com.
[6] S. Y. Wang and H. T. Kung, "A Simple Methodology for Constructing Extensible and High-Fidelity TCP/IP Network Simulators," *IEEE INFOCOM '99*, New York, Mar. 21–25, 1999.
[7] N. Gude *et al.*, "NOX: Towards an Operating System for Networks," *ACM SIGCOMM Comp. Commun. Rev.*, vol. 38, issue 3, July 2008.
[8] Floodlight OpenFlow controller, http://www.projectfloodlight.org/floodlight.
[9] T. R. Henderson, M. Lacage, and G. F. Riley, "Network Simulations with the ns-3 Simulator," *ACM SIGCOMM '08*, Seattle, WA, Aug. 17–22, 2008.
[10] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," *ACM Hotnets '10*, Monterey, CA, Oct. 20–21, 2010.
[11] B. Pfaf *et al.,* "Extending Networking into the Virtualization Layer," *Proc. HOTNETS 2009*.
[12] Open Networking Summit 2013, Santa Clara, CA, Apr. 15–17, 2013.
[13] IEEE 802.1AB, "Link Layer Discovery Protocol."
[14] R. Sherwood *et al.,* "FlowVisor: A Network Virtualization Layer," http://www.openflow.org/downloads/.../openflow-tr-2009-1-flowvisor.pdf.

Biographies

SHIE-YUAN WANG (shieyuan@cs.nctu.edu.tw) is a professor in the Department of Computer Science at National Chiao Tung University, Taiwan. He received his Master's and Ph.D. degrees in computer science from Harvard University in 1997 and 1999, respectively. He received the Outstanding Information Technology Elite Award of the Taiwan R.O.C. government in 2012, which was bestowed by the Vice President of Taiwan R.O.C..

CHIH-LIANG CHOU (clchou@estinet.com) received his Ph.D. degree from the Department of Computer Science, National Chiao Tung University. Now, he is the research director of EstiNet Technologies, Inc.

CHUN-MING CHAN (momo@estinet) received his Master's degree from the Department of Computer Science, National Chiao Tung University. He is an engineer with EstiNet Technologies, Inc.