NS4: Enabling Programmable Data Plane Simulation

Jiasong Bai, Jun Bi, Peng Kuang, Chengze Fan, Yu Zhou, Cheng Zhang
Tsinghua University

ABSTRACT

Network simulation plays a crucial role in the field of network research, education, and industry. However, before conducting a simulation on traditional network simulators, operators need to develop a simulative behavioral model, which requires intimate knowledge of the simulator implementation. Besides, the behavioral model cannot be migrated directly into real-world devices due to its tight coupling with the simulator platform, resulting in redundant and error-prone codes rewriting. Recently, P4, a high-level domain specific language (DSL), has attracted great attention from both academia and industry for its advantages of enabling operators to define behaviors of the programmable data plane.

Inspired by the idea of DSL, we present NS4, a P4-driven network simulator supporting simulation of P4-enabled networks to address the problems existing in traditional simulators. Taking advantage of P4, NS4 simplifies the development of a behavioral model and bridges the gap between simulation and deployment. Furthermore, to the best of our knowledge, NS4 is the first research effort to enable simulation of a P4-enabled network, providing a useful tool for P4 research and development. In this paper, we designed and implemented NS4, consisting of data plane models integrated with ns-3, the state-of-the-art network simulator, and control plane models to interact with the P4 pipeline. Then we evaluated its effectiveness and efficiency by simulating several representative P4 programs. Results show that NS4 can simulate large-scale P4-enabled networks at a low cost.

CCS CONCEPTS

• **Networks** → **Network simulations**; **Programmable networks**;

KEYWORDS

P4, programmable data plane, simulation

SOSR '18, March 28–29, 2018, Los Angeles, CA, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5664-0/18/03…$15.00
https://doi.org/10.1145/3185467.3185470

1 INTRODUCTION

Network simulation is widely used in nearly every aspect of network research, teaching, and industry [1] [2]. Through simulators, the behaviors of a real-world network get modeled and the corresponding results are presented. Since it is costly to deploy a testbed network, network simulators are typically used to validate and evaluate the design of network protocols and entities. To conduct such a simulation on a traditional network simulator (like ns-3), operators are required to follow the steps listed below: First, develop a behavioral model, which implements the design of network protocol or entity, as an internal module of the simulator. Second, set up the network topology and define tasks performed during the simulation. Third, trigger the simulation to test whether the behavioral model is behaving as expected. Finally, when verification is a success, the simulation codes need rewriting to get deployed in testbeds or vendor devices. During this process, traditional network simulators expose some significant **drawbacks**:

D1: Developing the behavioral model is time-consuming, error-prone, and even worse, requiring intimate familiarity with the simulator, which turns to be a steep learning curve.

D2: Tightly coupled with the specific simulator, simulation codes are difficult to be ported to real-world networks. For example, to bring a network entity into reality, it is inevitable to transform codes in a General Purpose Language (like C++) into those in a Hardware Description Language (like Verilog) or even Circuit Board Design. The transformation introduces both redundant work and potential bugs.

D3: Traditional network simulators lack support for the programmable data plane. Existing software switches supporting P4 (such as bmv2 [3] and pfpsim [4]) have to run with a network emulator (such as mininet [5]). A network ***emulator***, in contrast to a network ***simulator***, gets constrained by host resources, thus failing to simulate large-scale or ultra-speed P4-enabled networks.

The root of **D1** and **D2** can be concluded as the tight coupling between the implementation of the behavioral model and the simulator platform. We claim implementing a behavioral model should not require intimate knowledge of simulator implementation. Considering the need for simulators supporting programmable networks (**D3**), it is an appealing idea to integrate P4 into the network simulator.

In this paper, we present **NS4**, a P4-driven network simulator supporting simulation of large-scale P4-enabled networks. By introducing P4 into ns-3, the state-of-the-art simulation

Table 1: Comparison of ns-3, NS4.

Attribute	ns-3	NS4
Behavioral model development	C++	P4
Model decoupling	×	√
Direct migration	×	√
Programmable data plane	×	√

platform, NS4 decouples the behavioral model from underlying simulation platforms. As listed in Table 1, with the help of NS4, programmers only need to define behaviors of packet processors, without using any simulator-specific libraries (solution of **D1**). Moreover, the target independence of P4 allows behavioral models of NS4 to be migrated directly into real P4-enabled devices, which eliminates the redundant code rewriting (solution of **D2**). Furthermore, NS4 is a helpful tool which helps P4 researchers simulate and validate their works under arbitrary network contexts (solution of **D3**). With NS4, P4 researchers and engineers can efficiently evaluate various on-data-plane application designs, such as the distributed protocol accelerator [6], the Layer-4 stateful load balancer [7], the high-performance monitor [8], and the key-value store cache [9] in networks of arbitrary scales.

There are several challenges in integrating P4 into ns-3.

C1: Modeling behaviors of real P4 devices (e.g., Tofino [10] and P4FPGA [11]) based on ns-3 simulator.

C2: Existing runtime tools such as P4Runtime [12] and ONOS-BMv2 [13] for controlling P4 programs cannot interact with the simulated P4 module since ns-3 is a discrete-event simulator. The runtime operations have to be transformed into discrete events to get conducted.

C3: Simulating multiple P4 devices in a network requires installing routing entries in every switch. Since configuring flow entries manually is laborious and error-prone, automatic population of flow entries is needed to simulate networks with numbers of P4 devices.

NS4 builds an internal P4 pipeline to process packets as P4 programs defined, and enables queues and buffers in a pipeline to model behaviors of real P4 devices. To control the internal P4 pipeline during simulation, NS4 supplies several control modules to conduct runtime operations discretely. Besides, borrowing the idea of reactive flow rule population from OpenFlow [14], NS4 provides a module to compute and populate flow entries reactively to mitigate the laborious flow rule configuration.

Overall, this paper make the following contributions:

(1) Design and implementation of NS4, the first network simulator supporting simulation of P4-enabled networks, which is available on Github [15].
(2) Design of internal queues and buffers to model the behaviors of real devices, along with several control modules to enable runtime operations.
(3) An evaluation of effectiveness and efficiency of NS4 by simulating several representative P4 programs.

The rest of this paper is organized as follows. We provide background on P4 and ns-3 in Section 2. Section 3 describes the detailed design of NS4. The evaluation of NS4 is elaborated in Section 4. Section 5 introduces some related works. Finally, we conclude our work in Section 6.

2 BACKGROUND

In this section, we briefly provide a high-level overview of P4 language and ns-3 network simulator.

P4 language. P4 is a domain specific language for programming behavioral of programmable data plane architectures [16] [17], and is used to define how switches process the packets, A P4-enabled switch can be abstracted as a model containing a parser for extracting header fields, a collection of match-action tables that process these headers, and a deparser for reconstructing the packets. After receiving packets, the parser extracts fields from the header first and passes the fields to match-action tables. Each table matches specific fields configured by the controller, and perform the corresponding actions. Finally, the deparser reconstructs packet by writing the header fields back.

ns-3. ns-3 is a discrete-event network simulator in which the network system is modeled as events happen at discrete instances in time. There are several core concepts and abstractions in ns-3. Node is the fundamental computing device abstraction, representing the device connecting to a network. Application is the abstraction for a user program that generates activities to be simulated. Channel is the basic communication subnetwork abstraction. To simulate a network system, some common tasks is required to be done, including creating Netdevices, allocating MAC addresses, installing NetDevices on Node, configuring the protocol stacks of Nodes, and connecting the NetDevice to Channels, similar to build a real-world network.

3 DESIGN

3.1 Architecture of NS4

Figure 1 shows the overall architecture of NS4. NS4 is divided into a data plane half which contains a module for modeling a P4-enabled switch, *NS4NetDevice*, and a control plane half composed of modules for controlling the data plane. Developers simulate a P4-enabled device by instantiating a *NS4NetDevice*, loading the P4 program, and populating flow entries from the control plane. To assure compatibility of NS4, *NS4NetDevice* can be connected to other *NS4NetDevices* or traditional network devices like routers or switches (omitted in the figure).

To control the data plane module of NS4, the control plane is needed. It seems possible to use external controllers to

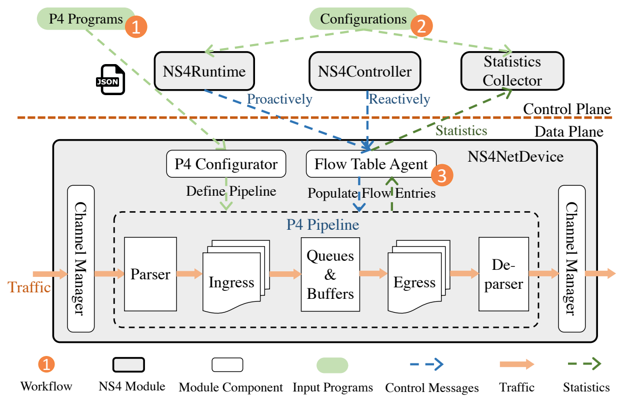


Figure 1: NS4 architecture.

control *NS4NetDevice*, since ns-3 supports interacting with external devices. However, we argue it is an inappropriate way since it makes NS4 a network emulator, which interacts with external entities and gets constrained by system resources, rather than a simulator. Besides, the coupling with external devices makes emulation results dependent on system resources, which means operators can receive different results in different devices with the same program. To enable P4 network simulation without introducing any external network entities, we create internal control plane modules to populate flow entries into flow tables and collect statistics from switches discretely. Consisting of flow table operations, user configurations are loaded and translated into discrete events as simulation begins, and get performed at the appointed time. To eliminate the laborious work of configuring routing entries, we also add a module in NS4 to enable reactive flow entries population during simulation.

The workflow of simulating a P4 network are shown as numbers in Figure 1: (1) Configure the behavior of data plane by inputting compiled P4 programs to P4 pipeline configurator; (2) Create control plane and configure the flow table operations and statistics collection tasks to be performed; (3) Build network topology, install applications and trigger the simulation. The control configurations are transformed into discrete events and get performed at the appointed time.

3.2 Programmable Data Plane of NS4

NS4NetDevice, designed for modeling P4 devices, is the basic module of the NS4 programmable data plane. By supplying interfaces for connecting with other network devices, *NS4NetDevice* enables operators to build a P4-enabled network consisting of multiple network devices. The basic goal of *NS4NetDevice* is to simulate the behaviors of the real-world P4 devices. We integrate a complete **P4 pipeline** into *NS4NetDevice* to simulate the behaviors of a P4 program. With the help of P4 compiler like p4c, NS4 could support all features and both two versions of P4. Moreover, to guarantee

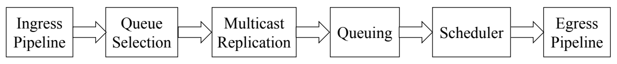


Figure 2: The packet flow through the P4 pipeline of NS4 data plane.

the fidelity of simulation and efficiency of simulation, we make a further design on the **queues** and **buffers**.

P4 Pipeline P4 pipeline is the core of *NS4NetDevice*. To enable customization of the pipeline, a configurator and a flow table agent are abstracted in the module. On the initialization of the device, the configurator loads the compiled P4 program to configure the behavior of pipeline. At runtime, the flow table agent can be called by control modules to add, modify, delete or query the entries of flow tables. Besides, we add a channel manager outside to conceal details of underlying channels and provide uniform interfaces for the pipeline, which expands the scope of simulatable P4 devices of NS4.

Queuing System. There are multiple queues in the module *NS4NetDevice*, and each egress port of the device can be associated with several queues. A queuing system is implemented to maintain and schedule the output queues. The packet flow through the queuing system is abstracted as Figure 2. When a packet leaves the ingress pipeline, it is accompanied by a metadata which determines actions to be taken in a queuing system, such as the set of ports to which the packet will be sent. According to the metadata, the corresponding set of queues is selected for each packet. A packet may get replicated and be sent to multiple ports for purposes like traffic flooding. After that, packets are moved to the selected queues and wait to be scheduled. The scheduler examines all queues which are eligible to transmit a packet, dequeues and passes the packet to the egress pipeline. Different priorities are assigned to queues, queues with a higher priority always get scheduled prior to those with a lower priority. For queues with the same priority, a weighted round robin (WRR) mechanism is adopted to choose the scheduled queue.

3.3 Control Plane of NS4

There are two main goals of the control plane in NS4: (1) Translate the user configurations into discrete events and trigger events at the appointed time; (2) Compute routing paths and populate flow entries reactively for packets failing to be matched in the P4 pipeline. *NS4Runtime* and *Statistics Collector* are responsible for the former while *NS4Controller* module accomplishes the latter.

Discrete operations. The user configurations are loaded into control modules during device initialization and get parsed into discrete events by control modules. Every line in configurations corresponds to a discrete event, consisting of a timestamp, a device id, a description of the action type and the corresponding parameters, as shown in table 2.

Table 2: NS4 event format.

Time stamp	Device ID	Command Type	Parameters

Table 3: NS4 flow table commands.

Command Type	Description	Parameters	Executor
table_set_default	Set default entry in a match table	<table name> => <action>	**NS4Runtime**
table_add	Add entry to a match table	<table name> <match fields> => <action>	
table_delete	Delete entry from a match table	<table name> <entry handle>	
table_delete_wkey	Delete entry using the match key	<table name> <match fields>	
table_modify	Modify entry in a match table	<table name> <entry handle> => <action>	
table_modify_wkey	Modify entry using the match key	<table name> <match fields> => <action>	
table_dump	Dump all entries in a match table	<table name>	**Statistics Collector**
counter_read	Read value(s) from counter	<counter name> <index | entry handle>	
counter_reset	Reset values for counter to 0	<counter name> <index | entry handle>	

The timestamp determines when the event to be performed and the device id specifies the targeted device. Commands type along with the parameters supported by NS4 are listed in Table 3. Events of adding, modifying and deleting table entries are performed by *NS4Runtime* while other events of dumping the tables and reading the counters are performed by *Statistics Collector*.

To record and schedule the events, *NS4Runtime* and *Statistics Collector* maintains an internal queue respectively. Events are arranged in the order of execution time in each module and get dequeued once the simulation time meets the appointed time.

Reactive population. It is a laborious work to manually configure all the forwarding rules, especially in a large-scale network. Fortunately, there are some standard routing rules to be installed in different network simulation scenes, such as rules to complete ARP process. Given network topology, these rules can be computed and populated during simulation. Borrowing the idea of reactive flow entry population from OpenFlow, we design *NS4Controller* module to populate flow entries reactively during simulation. Similar to the packet-in mechanism in OpenFlow, default action of every flow table is set to send the packet to *NS4Controller*. After receiving packets, *NS4Controller* computes routing paths according to the source and destination address of packets and populate the flow entries.

Since the flow tables in P4 devices are defined by P4 programs, the format of flow entries and name of the targeted table remain unknown to *NS4Controller*. Thus, operators need to pass the format and table name of each switch to *NS4Controller* before conducting a simulation. *NS4Controller* module is designed to be optional since it brings an extra burden on the system. Operators are able to choose whether to use the module according to the scale of simulation network.

4 EVALUATION

In this section, we explore the effectiveness and efficiency of NS4. We first evaluate the ability of NS4 to simulate a large P4-enabled network by simulating SilkRoad, a P4 load balancer for data center networks, in a fat-tree [18] network based on NS4. Then, we compare the development complexity of NS4 with original ns-3 by developing behavioral models of several representative data plane network functions. Finally, we evaluate the performance of NS4 along two dimensions: (1) Resource utilization; (2) Execution time. Our experiments are conducted on a Dell R730xd PowerEdge server [19] with two Intel Xeon-2620 CPUs, 64GB RAM and a Gigabit NIC.

4.1 Case Study

To illustrate the effectiveness of NS4, we simulate SilkRoad based on NS4 as a case study. In cloud data centers, a lot of traffic comes with a virtual IP address(VIP) and needs to be mapped to a pool of servers with direct IP addresses(DIP) by load balancing function [20]. While software load balancer(SLB) finishes the address mapping with a high overhead, SilkRoad enables load balancing to be performed at wire speed by offloading load balancing function to switches. However, since the lack of a P4-enabled simulator, it was not possible to evaluate the performance of SilkRoad in a data center network by simulating its behaviors until NS4.

To explore the performance of SilkRoad with NS4, we prototype and deploy SilkRoad in a fat-tree network with k equals 8. To make a comparison, we also implement an SLB application on the ns-3 platform. We use a network with k equals 4 to give a brief overview of the simulation scenario in Figure 4. To model data center traffic, we generate inbound traffic with VIPs following an on-off pattern with exponential random distribution [21], and send it to the network through core switches. Hosts connected to the same switch are considered as a pool of servers with DIPs, providing specific service for inbound packets. After receiving

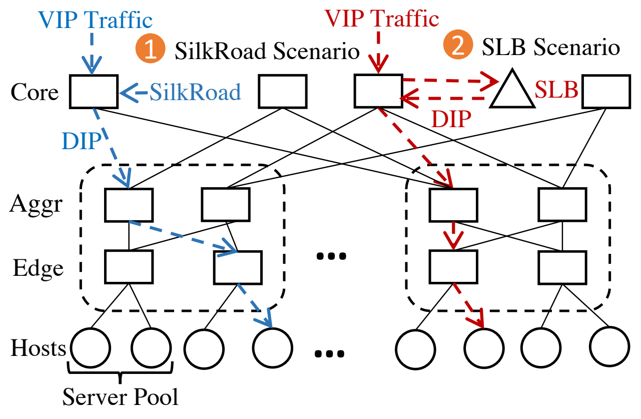


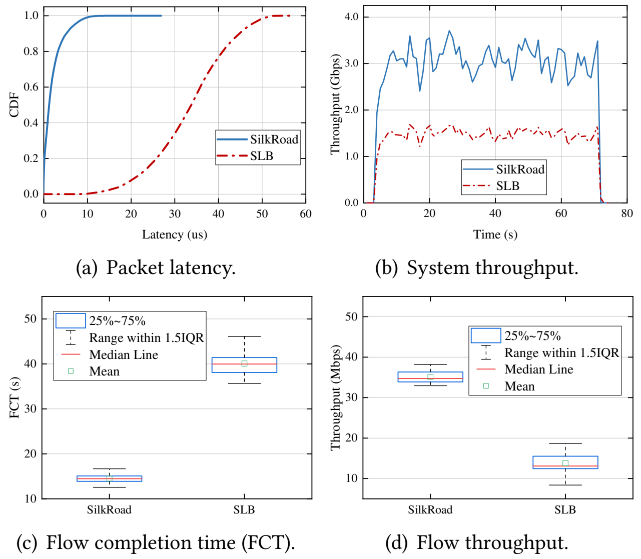
Figure 3: Simulation network.

packets with VIPs, switches deployed with SilkRoad conduct the address mapping itself (Scenario 1 in Figure 4) while the bare metal switches have to forward the packets to and fetch the packets with DIPs from SLB (Scenario 2 in Figure 4). After getting DIPs of packets, switches send packets to corresponding pools. We conduct simulations in two scenarios: (1) Deploy SilkRoad in every core switch (2) Connect two SLBs to every core switch. During simulation, we measure the performance of system along latency and throughput in both packet terms and flow terms.

The Cumulative Distribution Function (CDF) diagrams of packet latency in two scenarios are shown in Figure 4(a). Benefited from the high performance of switches deployed with SilkRoad in packets processing, packets in SilkRoad scenario have a smaller latency than those in SLB scenario. Figure 4(b) depicts the system throughput in two scenarios. In both scenarios, system throughput increases as the simulation begins and decreases to zero before simulation ends. During simulation, the throughput of SilkRoad system always outperforms that of SLB system since ASICs process packets much faster than SLBs. Completion time and throughput of data flows in two scenarios are depicted in Figure 4(c) and Figure 4(d). SilkRoad provides a direct path between inbound packets and targeted servers, eliminating the need for in-between SLBs. Thus, TCP flows in SilkRoad scenario receive responses faster than flows in SLB scenario, resulting in the difference in completion time and flow throughput. The above results indicate that SilkRoad has a better performance than traditional SLBs. This case shows that NS4 allows researchers to simulate their P4 programs on a large-scale network for verification and evaluation.

4.2 Development Complexity

We evaluate the development complexity of NS4 with several representative P4 applications. To make a comparison, we also implement these applications on the ns-3 platform. Since the headers and parsers of P4 programs can be shared and the



(a) Packet latency. (b) System throughput.

(c) Flow completion time (FCT). (d) Flow throughput.

Figure 4: Metrics measured in SilkRoad scenario and SLB scenario.

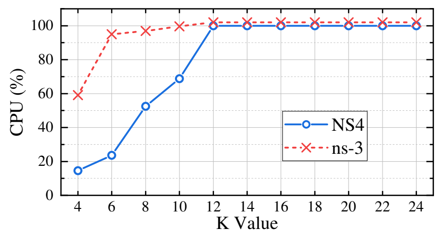
development of them is a one-time cost, we only take codes of match-action tables and control flows into account. Our first application implements the Layer 2 / Layer 3 forwarding (L2/L3 Switch) for packets. Based on it, the following applications accomplish several network functions including Access Control List (ACL) to filter out specific packets, Network Address Translation (NAT) to map IP addresses into another address space, Source Guard (SG) to filter out malicious packets on a Layer 2 port and Storm Control (SC) to prevent LAN ports from disrupted by broadcast.

NS4 reduces the difficulty and workload of behavioral model development significantly. Taking advantage of the device independence of P4, NS4 enables operators only write the packet processing logic codes without calling any internal libraries of the simulator. As listed in Table 4, line of P4 applications implementing the network functions is less than half of line of codes of C++ applications.

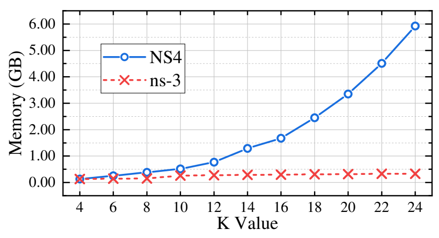
4.3 Simulation Performance

To explore the ability of NS4 to simulate large-scale networks, we simulate fat-tree networks with different k values with NS4. As for network traffic, we randomly select communication pairs including a sender and a receiver across all hosts. In the simulation, the sender in each pair sends 1 Mbps traffic to the receiver simultaneously. Before conducting the simulation, we pre-calculate and populate routing flow rules to each switch. We set simulation seconds as 100 seconds each time and measure the performance of simulator along resource utilization and execution time.

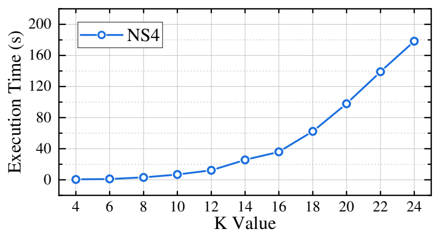
Resource Utilization. We select CPU and memory as metrics to measure the resource utilization. The results are shown in Figure 5(a) and Figure 5(b). As k value increases



(a) CPU utilization.



(b) Memory utilization.



(c) Execution time.

Figure 5: NS4 performance.

Table 4: Behavioral model development complexity of ns-3 and NS4.

Cases	Features	ns-3	NS4
#1	L2/L3 Switch	598	165
#2	L2/L3 Switch, ACL	803	252
#3	L2/L3 Switch, ACL, NAT	1038	494
#4	L2/L3 Switch, ACL, NAT, SG, SC	1219	637

from 4 to 24, CPU utilization of ns-3 rises faster than that of NS4. Memory occupation of NS4 increases from 131 MB to 6008 MB while memory occupation of ns-3 remains no more than 400 MB. This is because the different routing strategies adopted by two simulators. ns-3 adopts an on-demand routing policy in looping networks, calculating routes for packets during the simulation, which saves memory resources but consumes more CPU resources and requires a longer execution time (even hours). In contrast, NS4 pre-computes and populates routing entries in every switch, resulting in a lower CPU utilization, a shorter execution time (minutes) but an $O(k^3)$ memory occupation. Since not all entries will get used in the simulation, operators can populate part of entries to save memory.

Execution Time. As depicted in Figure 5(c), execution time rises from 4 seconds to 178 seconds as k value increases. The execution time of ns-3 is omitted in the figure since it is at the hour level. NS4 is outperforming ns-3 by times in execution time because NS4 does not need to compute routing rules during simulation, with all forwarding rules are calculated in advance. Thus, the simulation can be completed in minutes even there are thousands of hosts in the network, which allows NS4 simulation to be conducted on hosts with limited computation resources.

5 RELATED WORKS

Our work is motivated by both previous network simulators and P4 toolkits, along with some industry P4 technologies.

Network Simulators. OPNet [22] is a commercial network simulator based on discrete events. Besides development environment for specification of the simulation, it also provides a graphical GUI for the design of the simulation [23]. ns, consists of ns-1, ns-2 [24] and ns-3 [25] is a series of discrete-event open source network simulators developed by different groups and organizations. ns-3, first released in 2008, has been widely used in research and teachings, and a lot works [26] [27] [28] has been done to extend ns-3 to support more network scenarios. PFPSim provides a simulator for programmable forwarding plane devices, which enables operators to define the architecture and processing behavior of device by high-level languages including C++ and P4. PFPSim supplies a software switch model which can load P4 programs, but it still lacks some components to simulate a complete P4-enabled network.

P4 Toolkits. P4.org [29] offers a set of toolkits to help compile, verify, and run P4 programs, including bmv2 which provisions a standard behavioral model for P4 language, p4c, a reference compiler for the P4 programming language, and P4 Runtime which provides a control plane framework and tool for P4. Besides, there are works about P4 compiling in both academia and industry [30]. Barefoot Networks provides Capilano software development environment [31] to compile P4 programs into their Tofino chips. Introducing P4 into network simulation, our work is demonstrated on [32].

6 CONCLUSIONS

In this paper, we proposed NS4, a discrete-event network simulator for modeling a network containing one or more P4-enabled devices. By introducing P4 into traditional network simulators, NS4 not only simplifies the development of behavioral model but also enables the simulation codes to be migrated directly to real-world devices. To the best of our knowledge, NS4 is the first simulator supporting large-scale P4-enabled networks, which enables researchers to validate and evaluate their P4 programs. We implement and evaluate NS4 and the results shows NS4 is able to simulate a P4-enabled network with acceptable overhead.

ACKNOWLEDGEMENT

This research is supported by National Key R&D Program of China (2017YFB0801701), the National Science Foundation of China (No.61472213) and CERNET Innovation Project (NGII20160123). Jun Bi is the corresponding author. We are grateful to anonymous reviewers and shepherd, Anirudh Sivaraman for valuable and constructive comments that helped us improve the manuscript.

REFERENCES

[1] ns-3. https://www.nsnam.org/, 2011.

[2] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, et al. Advances in network simulation. *Computer*, 33(5):59–67, 2000.

[3] Behavioral model. https://github.com/p4lang/behavioral-model.

[4] Samar Abdi, Umair Aftab, Gordon Bailey, Bochra Boughzala, Faras Dewal, Shafigh Parsazad, and Eric Tremblay. Pfpsim: A programmable forwarding plane simulator. In *Architectures for Networking and Communications Systems (ANCS), 2016 ACM/IEEE Symposium on*, pages 55–60. IEEE, 2016.

[5] Mininet Team. Mininet: An instant virtual network on your laptop (or other pc). http://mininet.org/, 2012.

[6] Jialin Li, Ellis Michael, Naveen Kr. Sharma, Adriana Szekeres, and Dan R. K. Ports. Just say no to paxos overhead: Replacing consensus with network ordering. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 467–483, GA, 2016. USENIX Association.

[7] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *Proceedings of the 2017 ACM SIGCOMM Conference*, SIGCOMM '17, pages 525–538, Los Angeles, CA, USA, 2017. ACM.

[8] Mojgan Ghasemi, Theophilus Benson, and Jennifer Rexford. Dapper: Data plane performance diagnosis of tcp. In *Proceedings of the Symposium on SDN Research*, SOSR '17, pages 61–74, New York, NY, USA, 2017. ACM.

[9] Jin Xin, Xiaozhou Li, Zhang Haoyu, Robert Soule, and Jeongkeun. Lee. Netcache: Balancing key-value stores with fast in-network caching. P4 workshop 2017. http://p4.org/wp-content/uploads/2017/06/p4-ws-2017-netcache.pdf.

[10] Barefoot Networks. Tofino. https://barefootnetworks.com/technology/.

[11] Han Wang, Robert Soulé, Huynh Tu Dang, Ki Suh Lee, Vishal Shrivastav, Nate Foster, and Hakim Weatherspoon. P4fpga: A rapid prototyping framework for p4. In *Proceedings of the Symposium on SDN Research*, SOSR '17, pages 122–135, New York, NY, USA, 2017. ACM.

[12] P4runtime: a control plane framework and tools for the p4 programming language. https://github.com/p4lang/PI.

[13] P4 brigade. https://wiki.onosproject.org/display/ONOS/P4+brigade.

[14] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[15] Ns4. https://ns-4.github.io, 2017.

[16] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 99–110, New York, NY, USA, 2013. ACM.

[17] Sharad Chole, Andy Fingerhut, Sha Ma, Anirudh Sivaraman, Shay Vargaftik, Alon Berger, Gal Mendelson, Mohammad Alizadeh, Shang-Tse Chuang, Isaac Keslassy, Ariel Orda, and Tom Edsall. drmt: Disaggregated programmable switching. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 1–14, New York, NY, USA, 2017. ACM.

[18] Charles E Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE transactions on Computers*, 100(10):892–901, 1985.

[19] Dell. Poweredge r730 rack server. Website. http://www.dell.com/en-uk/work/shop/productdetailstxn/poweredge-r730.

[20] Parveen Patel, Deepak Bansal, Lihua Yuan, Ashwin Murthy, Albert Greenberg, David A Maltz, Randy Kern, Hemant Kumar, Marios Zikos, Hongyu Wu, et al. Ananta: Cloud scale load balancing. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 207–218. ACM, 2013.

[21] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. *ACM SIGCOMM Computer Communication Review*, 40(1):92–99, 2010.

[22] Xinjie Chang. Network simulations with opnet. In *Simulation Conference Proceedings, 1999 Winter*, volume 1, pages 307–314. IEEE, 1999.

[23] Sebastian Rampfl. Network simulation and its limitations. In *Proceeding zum Seminar Future Internet (FI), Innovative Internet Technologien und Mobilkommunikation (IITM) und Autonomous Communication Networks (ACN)*, volume 57, 2013.

[24] Kevin Fall and Kannan Varadhan. The network simulator (ns-2). *URL: http://www. isi. edu/nsnam/ns*, 2007.

[25] Thomas R Henderson, Mathieu Lacage, George F Riley, C Dowell, and J Kopena. Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, 14(14):527, 2008.

[26] Hemin Yang, Chuanji Zhang, and George Riley. Support multiple auxiliary tcp/udp connections in sdn simulations based on ns-3. In *Proceedings of the Workshop on ns-3*, pages 24–30. ACM, 2017.

[27] Steven Smith, David R Jefferson, Peter D Barnes Jr, and Sergei Nikolaev. Improving per processor memory use of ns-3 to enable large scale simulations. In *Proceedings of the 2015 Workshop on ns-3*, pages 60–66. ACM, 2015.

[28] Sérgio Conceição, Filipe Ribeiro, Rui Campos, and Manuel Ricardo. Novel ns-3 model enabling simulation of electromagnetic wireless underground networks. In *Proceedings of the 2015 Workshop on ns-3*, pages 9–16. ACM, 2015.

[29] Behavioral model repository. https://p4.org/.

[30] Lavanya Jose, Lisa Yan, George Varghese, and Nick McKeown. Compiling packet programs to reconfigurable switches. In *NSDI*, pages 103–115, 2015.

[31] Barefoot Networks. Barefoot capilano. Website. https://barefootnetworks.com/products/brief-capilano/.

[32] Chengze Fan, Jun Bi, Yu Zhou, Cheng Zhang, and Haisu Yu. Ns4: A p4-driven network simulator. In *Proceedings of the SIGCOMM Posters and Demos*, pages 105–107. ACM, 2017.