NS4: Enabling Programmable Data Plane Simulation

Meister Rados

Abstract

A new idea of forwarding packets within a Software-Defined-Network is the programmable data plane. Its innovative inherent protocol independence has attracted attention of networking operators recently. However, state of the art network simulators lack support for simulating large scale networks with programmable data plane. In this paper, NS4 is presented. It is an open source network simulator and addresses the aforementioned issue by allowing simulation of the programmable data plane. Furthermore, the workflow of conducting simulation is optimized. Cumbersome steps that used to arise are automated. Benchmarks in this paper show, that NS4 outdoes its predecessor, ns-3, by significant factor, regarding capability, speed and code density.

1 Introduction

Software-Defined-Networking (SDN) provides, compared to traditional networks, more advantages to their operators regarding management complexity and its costs, controllability, flexibility in configuration and more. The next step in moving towards independence of traditional networking constraints is to change the forwarding potion within SDN. This can be achieved by making use of the programmable data plane, an idea that has emerged from creating innovative ways to manage network's forwarding behavior. The programmable data plane attractive to network operators, since it can be reconfigured even after deploying the network devices. This leads to protocol independence and overall flexibility of the forwarding behavior without touching any part of the physical network or rebuilding its topology. The programming language used to describe the forwarding behavior is P4, which will be presented in more detail later on. In this paper, NS4, the first simulation environment for multiple P4 enabled network devices is presented.

The structure of this discussion is as follows: First, some background information is given and related work is presented. This is followed by a motivation. Then, NS4 itself and its technical inner workings are regarded. Benchmarks and performance evaluation close this discussion.

This discussion is based on the work done by [BBK+18]. Consequently, a significant portion of this paper is adapted from the original work.

2 Background Information

In this section, an explanation of programmable data plane is given, the difference between simulation and emulation is outlined and the concept of P4 will be explained.

2.1 Programmable data plane

The programmable data plane is a forwarding idea that belongs to software defined networking (SDN). The concept of the programmable data plane is simple: Operators can describe with software, how networking devices process network traffic. These devices can be reconfigured at runtime by loading other configurations into the networking device. This brings advantages over traditional forwarding, for example protocol independence or implementation of arbitrary network functions like a firewall or network address translation in a single device.

2.2 Simulation and Emulation

In subsection 2.3, further network simulators are presented, which are available alongside with NS4. Among these, some might have rather characteristics of an emulator, which will explicitly be stated. Since the terms “Simulation” and “Emulation” have different meanings, it is important to what extend they differentiate and to know how to tell them apart.
Here are the major differences:

- Purpose: A simulation is conducted mainly for analytical purposes. Its results may server for research and development. Unlike that, an emulation is often used to logically imitate the behavior of a system, whose hardware isn’t available, but can run (be emulated) on another system.

- Interactivity: An emulation is designed to be interactive. It has input signals that can be set by an external resource and trigger events to which the emulated system responds. This can be a human or another system. On the other side, a simulation is a closed system. There are no such inputs from external resources. Events occurring at runtime are determined by a fixed simulation script. Therefore the occurrence of all events are theoretically known before simulation begins.

- Reproducibility: Since a simulation follows a fixed script, the behavior of a given system is always the same when running the same script. This assures full reproducibility. Whereas an emulation not necessarily follows a script, but can react to unforeseen external inputs, behavior of the emulated system may vary significantly.

- Runtime speed: The emulation speed is restricted to real world time. So, one emulated second is equal to one second of real time. This restriction is due to the property, that the real world can be an input for the emulated system. In contrast, a simulation can be speed up until the hardware resources of the executing platform are fully utilized.

- Precision: A simulation can have multiple levels of precision, depending on the abstraction of the simulated entity. If the underlying system is regarded downright as a white box, simulation results are expected to be precise at the cost of simulation speed. Operators have the option to add more abstraction to the simulated entity, facing a trade off between precision and simulation speed. An emulation has only two levels of precision. The emulated behavior can either be correct, or wrong.

2.3 Related Work

Besides NS4, there are some established commercial network simulators. On some of them, research is done to also enable programmable data plane simulation. A list of most related work is given:

1. ns-1 is a terminal based network simulation tool, developed by the Network Research Group at the Lawrence Berkeley National Laboratory [Lab97]. It is also known under the name LNBL Network Simulator and is the first version of the open source network simulator series ns. It evolved as a variant from the REAL Network Simulator [Kes97] in 1989. The development was supported by DARPA through the VINT project at LBL, Xerox PARC, UCB, and USC/ISI [Ins]. It supports simulation of a couple implementations of TCP, like Tahoe or Reno, as well as router scheduling algorithms. The used programming language is C++. Simulation scripts are written in Tcl [com18]. Tools like Gnuplot [WK18] are used to visualize the simulation results.

2. ns-2 is the successor of ns-1. Its development was supported by DARPA with SAMAN and through NSF with CONSER, both in collaboration with other researchers including ACIRI [HC01]. In contrast to its predecessor, it uses OTcl instead of Tcl. OTcl is an extension for Tcl, introducing a dynamic object and class model by using solely the C-API of Tcl. The interface code to the OTcl interpreter is separate from the main simulator. More complex objects from ns-1 have been decomposed to simpler components to ensure greater flexibility and support composing [UBP11].

3. ns-3 is the successor of ns-2, written in C++ and offering support for Python scripts instead of OTcl. It focuses on real world circumstances such as sockets, IP addresses, network devices and multiple interfaces per node. It is designed to be easier to incorporate with other open-source networking software, like kernel protocol stacks, packet trace analyzers and routing daemons while adhering to several C++ design patterns like templates, smart pointers, copy on demand and callbacks. On top of that, ns-3 offers a tracing system for customization of the statistics output without the need of restructuring the simulation core. A new testbed integration, support for virtualization and an attribute system eases the simulation progress even more [RHLFR08]. As its predecessors, ns-3 is has no GUI and is a discrete, event driven simulator. It is licensed under GNU GPLv2 [Fou17] and developed by its community [nP18].

4. OPNet was a commercial network simulation tool, founded in 1986 by Alain Cohen and Marc Cohen. It was bought by Rivernet Technology Inc. for a total of $1,016,000,000 in 2012 and has since then been further developed [Inc]. Unlike NS4, it offers a graphical user interface, but still lacks support for programmable data plane.

5. PFPSim is similar to NS4 a open source network simulator [Gro] which is also able to simulate programmable data plane [AAB$^+$16]. But it is not yet possible to use it for simulating large scale P4 enabled networks, since it is not capable of simulating multiple instances of P4 devices.

6. Mininet is an emulator for deploying large networks on the limited resources of a simple single computer or virtual machine. Mininet has been created for enabling research in Software Defined Networking (SDN) and OpenFlow [KSG14]. It offers no support for programmable data plane by itself, but theoretically could, with the help of p4lang's special software switch, called behavioral model version2 (bmv2) [p4l]. Even with that feature implemented, Mininet remains a network emulator, rather than a network simulator.

7. CrystalNet is a high fidelity, cloud-scale emulator developed by Microsoft Azure and Microsoft Research teams. Its purpose is to emulate the same topology, software configuration and hardware that is used in Microsoft's cloud architecture, so that operators can find potential problems proactively. [Bah17]. It is not capable of simulating P4 enabled devices and in contrast to NS4, it is a network emulator, not a network simulator [LZP$^+$17].

2.4 P4: Programming protocol-independent packet processors

P4 is a high level, domain specific programming language for describing behavior of protocol independent packet processors. It is used to define how switches process the packets, no matter what type underlying hardware is: CPU, FPGA, ASIC, etc [BDG+14]. P4 is used to make up the programmable data plane. A networking device is called "P4 enabled", once it has the capability to run P4 code. The architecture basically consists of three major parts:

1. Parser for extracting fields from the header of an arriving packet
2. Match-action tables that contain information how to process a packet, depending on the extracted header information
3. Deparser to assign new header files to a packet before it finally leaves the switch

Once a network simulator can run P4 code, programmable data plane simulation is achieved.

```
table routing {
   key = { ipv4.dstAddr :  lpm; }
   actions = { drop; route; }
   size :  2048;
}
control ingress() {
   apply {
      routing.apply();
   }
}
```

Figure 1: P4 sample code [Con18]

3 Motivation

Development processes of features and improvements in technology are well interwoven by simulation in order to validate functional correctness [ML01]. The current state-of-the-art open-source network simulator, ns-3 [RH10], is not capable of the programmable data plane. Since building a physical test environment with programmable date plane is impractical due to expensive devices, complex wiring, requiring of high expertise and possible deployment issues, a P4 enabled simulator is urgently needed. Therefore, researchers created NS4, a network simulator that is capable of simulating programmable data plane. In addition to that, it also solves more issues that came along when conducting a network simulation with ns-3. Having a look at the workflow of a simulation using ns-3, some arduous steps come up: Operators first need to develop a simulator specific behavioral model that implements the network and protocol design. This requires intimacy with the simulator and its libraries, which operators need to achieve at first. Flow entries then need to be installed by hand, a time-consuming and error-prone process that could be automated. The resulting behavioral model is tightly coupled with this particular simulator and therefore doesn't offer portability to neither other simulators nor to a physical real-world device. This leads to code re-writing, from a high level programming language to a hardware description language, a time consuming and again error prone process. Researchers solved these problems by creating the urgently needed successor of ns-3: NS4 [FBZ+17]. The main features that are improved are given in table 2.

Problem	**ns-3**	**NS4**
Simulator dependent	Yes, library calls used	No
Migration to real devices	Only after code re-writing	Directly supported
Programmable data plane	Not supported	Inherent support
Modeling language	C++	P4

Figure 2: Feature comparison of ns-3 and NS4. Source: [BBK$^+$18]

4 NS4 architecture

This section gives an overview of NS4's architecture. The function and role of its components are pointed out. After that, the control plane is regarded. The two options that operators have to populate flow table entries are explained. Lastly, a typical workflow when conducting a simulation with NS4 is described.

4.1 Overview

The architecture of NS4 is divided into a control plane and a data plane. The former is used for controlling the latter. Further details of the control plane are being pointed out in section 4.3. The interesting thing is the data plane, in which operators may instantiate and simulate "NS4NetDevice", a P4 enabled networking device. This enables simulation of the programmable data plane. Operators can load a compiled P4 program into the P4 enabled device and populate flow table entries from the control plane.
Furthermore, NS4 is capable of simulating other, non-P4-devices, since it is seamlessly downward compatible with ns-3. This assures compatibility with traditional devices. However, when it comes to NS4 benchmarks in the later sections, this feature will not be regarded.
To avoid instantiating of further resource-intensive external devices, like a controller, every NS4NetDevice has its own internal control plane module, which can install flow entries proactively and reactively. A channel manager serves for abstraction and offers the possibility to connect multiple "NS4NetDevice"s to build up a network topology.
A statistics collector is used to query information with statistical purpose from "NS4NetDevice".

4.2 NS4NetDevice

The programmable data plane is brought into simulative environment via the networking device "NS4NetDevice". This device is the P4 enabled equivalent to the "NetDevice" used in ns-3. Its inner structure is depicted in figure 3. This subsection explains how operators can build a network with multiple instances of "NS4NetDevice" and describes the internal packet processing done by the P4 pipeline.

4.2.1 Building Topology

Operators can build their network topology by connecting multiple instances of these devices. In order to provide uniform communication interfaces in between the simulated devices, "NS4NetDevice" has an channel manager implemented, that abstracts the input and output ports as well as hides complexity of internal mechanisms from the operator. The channel manager passes incoming packets to the P4 pipeline parser while checking for packets that have left the deparser and wait to be sent. These packets are then send out at the corresponding port.

4.2.2 Packet processing

Besides the channel manager, “NS4NetDevice” has an ingress pipeline and an egress pipeline to accomplish packet processing. On top of that, it holds an queuing system in between that decouples the two pipelines from each other. The packet flow can be described by the following steps:

1. Packet arrives at the switch
2. Channel manager passes it at the appointed time of simulation to the parser
3. Packet disassembly into header fields and payload
4. Ingress pipeline applies P4 described actions to the header fields
5. Accompanying header fields by metadata for queue selection
6. Replication of packet, if designated
7. Queue selection and enqueuing of packet
8. Scheduler dequeues packet at scheduled time
9. Egress pipeline applies P4 described actions to the header fields
10. Deparser reassembles the header files and the payload to a packet
11. Channel manager sends packet out at corresponding port

The packet flow through “NS4NetDevice” is depicted in figure 3.

4.2.3 Queuing System and scheduler

NS4NetDevice holds a queuing system and makes use of a scheduler. The former decouples the ingress- from the egress pipeline. A scheduler algorithm then detects whether the queue holds a packet that is ready and appointed to be sent before passing it to the egress pipeline. This queuing system leads to maintenance and schedule for the output queues. The scheduler algorithm can handle priorities, which ensures QoS support [MM16]. In case the priorities are equivalent, uses a weighted round robin mechanism, a commonly used policy when it comes to load balancing [WC14] or a fair distribution [IKM15].

4.3 Control Plane of NS4

The NS4 architecture is completed by the control plane. It consists of three modules: “NS4Runtime”, “NS4Controller” and “Statistics Collector”. The first module, “NS4Runtime”, is responsible for translating user configurations that are not for statistical purpose into discrete time events. To do so, it reads the configuration line by line and creates a discrete event consisting of a timestamp, an device identifier, a command type identifier and the specified parameters. The timestamp indicates simulation time at which the event needs to be performed. The device identifier specifies, which device will execute the event at hand. The command type identifier gives information on what kind of command should be executed. Available commands are depicted in table 4. The commands are passed to the control module in the “NS4NetDevice”, which populates its flow tables accordingly. Therefore, authors of [BBK$^+$18] call the flow table population that is derived from discrete time events “discrete

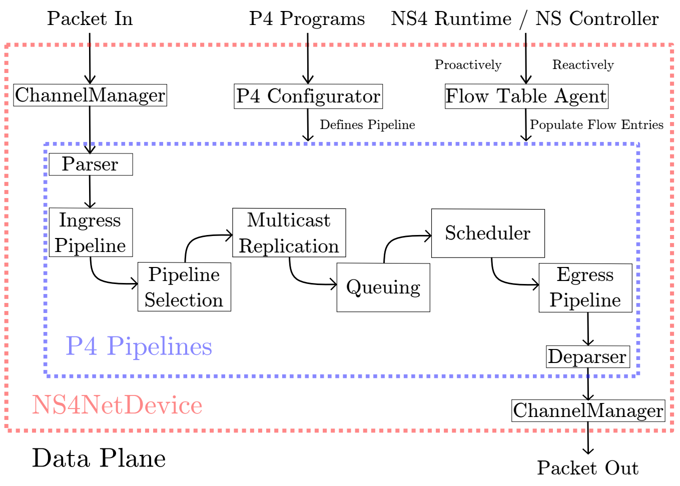


Figure 3: Architecture of NS4NetDevice. Adapted from: [BBK+18]

population". Configurations for statistical purpose are transformed into discrete events by the module "Statistics Collector". Supported actions are dumping all entries from a match table or reading and resetting specific counters.

Command Identifier	**Parameter List**	**Explanation**
table_add	table name, match fields, action	Create new entry in a table to specify actions to be performed when packets meet a match field
table_delete	table name, entry handle	Remove an entry from the match table
table_delete_wkey	table name, match fields	Remove a specified entry from the match table
table_modify	table name, entry handle, action	Modify an entry in a match table
table_modify_wkey	table name, match fields, action	Modify a specified entry in a match table
table_set_default	table name, action	Set the default action to perform for a match table

Figure 4: Available commands for "NS4Runtime". Adapted from [BBK+18]

The last module, "NS4Controller" allows operators to generate an automatic population. It is designed to be optional. Common and standard flow table entries can be derived automatically from the network topology by the "NS4Controller". This eliminates cumbersome manual population of trivial flow table entries. It also is capable of installing flows reactively. Unlike traditional controllers [HP15] that know all flow entries of the switches, the "NS4Controller" does not know about the flow table entries of the switches because they are derived from P4 programs that are stored inside every switch. This information needs to be passed manually to the "NS4Controller" to enable automatic population of flow table entries. This circumstance afflicts runtime performance, resulting a trade off between laborious work and simulation speed. As said, this module is designed to be optional. With the simulated network size in mind, operators may choose whether an instance of this module eases setting up a simulation significantly, or populating these flow entries by hand is feasible.

4.4 Workflow

The workflow of setting up a simulation with NS4 is as follows:

1. Describe programmable data plane behavior using P4 language. An integrated P4 compiler, p4c, compiles the P4 programs and passes them to the P4 pipeline
2. Set up the control plane by configuring flow table operations and determining when and which statistics information shall be gathered.
3. Install applications and define network topology

After these steps, simulation setup is done. When triggering it, the control configurations are transformed into discrete events and get performed at the appointed time.

5 Performance in application cases

The following section first presents the performance and effectiveness of NS4 in a case study, then shows the complexity growth measured in source code length compared to an equivalent implementation in ns-3 and eventually efficiency regarding resource utilization and execution time will be inspected. The simulations are executed on a Dell PowerEdge R370.

5.1 Silk Road: Example use case for NS4

In this subsection, the effectiveness of NS4 will be demonstrated by simulating a exemplary implementation of a P4 enabled network performing load balancing in data center networks. The next sections give some information on the characteristics of the implemented network as well as on the simulation setup.

5.1.1 Load balancing in data centers

In cloud data centers, incoming packets often belong to a virtual IP address and need to be mapped to a server with a direct IP address. In such scenario, arriving packets do not have a particular physical host as target. There are multiple equivalent servers online in the data center that can handle the request. A load balancing function takes care of equivalent route traffic among the possible routes and serve for prevention of server overloading [PBY+13]. This can be implemented in software using SLBs (Software Load Balancer). The SLB runs on a dedicated machine that is connected to the core switch of the data center network. The load balancing function can also be implemented , for example, by programmable switches that execute SilkRoad.

5.1.2 Silk Road and Simulation setup

SilkRoad [MZK+17] is an implementation of a Load Balancer written in P4, which can now be simulated with help of NS4. Without presence of NS4, the presented results (see section 5.1.3) were not available to the operators, allowing them now to evaluate performance and validate correctness of their P4 programs. For comparison of ns-3 with NS4, two scenarios are set up. Both have FatTree Network [Lei85] with k equals 4 (see figure 5) as topology. The difference is the way, in which the load balancing function is implemented. Both practically

achieve the same behavior, but in ns-3 scenario, the balancing task is done by 2 SLBs that are connected to the core switches (like shown in figure 6), whereas NS4 makes use of P4 enabled devices running Silk Road in every core switch (as shown in figure 7).
To generate seemingly real incoming VIP traffic, an off-pattern with exponential random distribution [BAAZ10] is generated. The main difference in the two implementations is that the traditional devices in ns-3 scenario need to pass all incoming packets to a SLB, which then decides where the packet should be forwarded to, to guarantee a well balanced load among all servers. On the other hand, Silk Road offloads this task to the P4 enabled devices, so they accomplish this task by themselves, leading to wire speed forwarding. Measuring throughput and latency of packets and flows shows that SilkRoad outperforms the SLB solution by far.

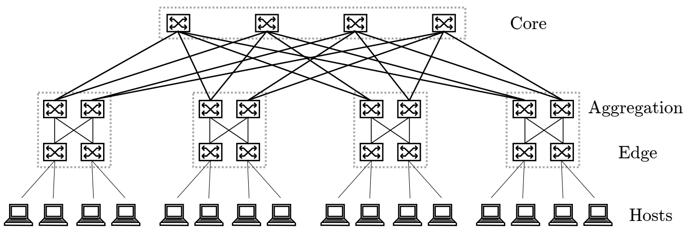


Figure 5: A fat tree network with k equals 4

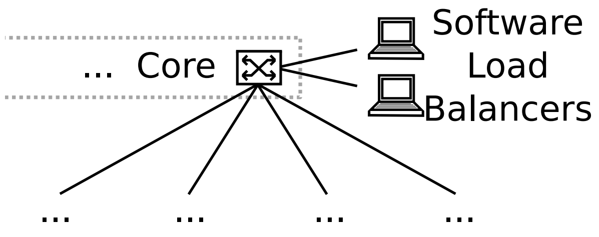


Figure 6: In ns-3 scenario, two SLBs are connected to each core switch

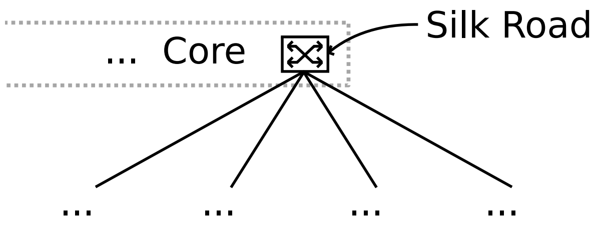


Figure 7: For comparison, NS4 simulates Silk Road, P4 load balancing run by all core switches

5.1.3 Evaluation of Silk Road Performance

In order to evaluate the aforementioned comparison, multiple metrics are used and their results plotted in figure 8. The cumulative distribution function (CDF) depicts the latency of the packets that were sent through the simulated network. For a given time *latency* the percentage of all packets having a latency smaller than *latency* is summed up and plotted in figure 8a). Regarding the graph, it is evident that all packets forwarded by Silkroad have a significantly smaller latency than about 80% of all packets forwarded by SLBs.
Figure 8b) plots the throughput of the simulated network over a fixed simulation time. As expected, with the results of 8a) in mind, the throughput of SilkRoad compared to the

implementation using SLBs, is much higher. Calculating the integral below these graph gives the amount of total packets processed and forwarded. Silk Road outperform traditional SLBs by a factor of approximately 2.
Figure 8c) shows the completion time of all flows that traveled through the simulated network. Not only a much shorter, but also more stable completion time can be observed.
Lastly, the flow throughput is plotted in figure 8d). Once again, Silk Road has a much higher throughput than load balancing done in software. All of these observations can be ascribed to the fact that Silk Road does not need to forward packets to a dedicated device that implements the load balancing in software. Silk Road is run directly in the P4 enabled core switches, that complete load balanced forwarding in hardware at wire speed.

Graphics removed due to copyright law

Figure 8: Comparison of SLB and Silk Road performance. Source: [BBK+18]

5.2 Code Complexity

In order to compare the code complexity of NS4 with ns-3, representative applications are being implemented in both simulators. Lines of code required for the implementation are summed up and serve as a metric for comparison. Headers and parses of P4 programs are excluded from the sum since they can be shared among the implementations. Only control flows and match-action tables are being counted. The first implementation is a basic Layer 2 / Layer 3 switch. Further features that serve for comparison are being implemented using that switch as a base implementation.
These features are as follows: Network Address Translation (NAT) that map IP Addresses from one address space into another; Source Guard (SC), a layer 2 function for filtering particular suspicious packets; Access Control List (ACL), another filtering algorithm and finally Storm Control (SC), that keeps LAN ports from being severed. Code complexity is given in table 9.

Features	**ns-3**	**NS4**	**Percent smaller**
Layer 2 / Layer 3 Switch	598	165	72.408%
Layer 2 / Layer 3 Switch & ACL	803	252	68.617%
Layer 2 / Layer 3 Switch & ACL & NAT	1038	494	52.408%
Layer 2 / Layer 3 Switch & ACL & NAT &SC & SG	1219	637	47.744%

Figure 9: Code complexity comparison: implemented features and the required number of lines of code. Source: [BBK+18]

Implementing features with P4 instead of C++ serves for device independence, there are no library calls needed. Describing only the logic of how switches process their packages increases simplicity in behavioral model development vastly. This leads to an average reduction of code length by 60.294%.

5.3 Demonstration of NS4 performance

The effectiveness of NS4 will be demonstrated by simulating a large scale P4 enabled network with multiple P4 devices. For comparison, the same network will be implemented in ns-3, using traditional switches, that aren't P4 enabled.

5.3.1 Simulation setup

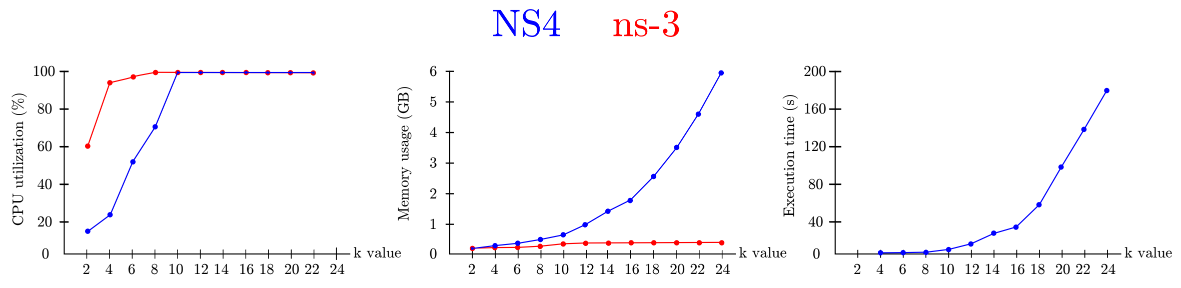
Likewise as in section 5.1, a Fat Tree Network is the topology of the simulated network. Multiple runs with k value increasing in steps of two from four to twenty-four are being conducted. Traffic is generated randomly, but underlies three constraints:

1. Sender and receiver are chosen randomly, but cover all hosts in the network
2. Every flow of packets between sender and receiver has a size of 1 Mpbs.
3. All flows start to send at the same time

The optional module “NS4 Controller” is used to enable automatic population of flow table entries in the switches. Time taken to generate automatic population is not being taken into account when evaluating simulation performance of NS4. The network is simulated for 100 seconds.

5.3.2 Hardware utilization

While the simulation is running, CPU and memory usage are monitored and serve as metrics for hardware utilization. Needlessly to say, simulation time is also being measured.
Figure 10a) shows that CPU usage for running the simulations with NS4 increases slower than with ns-3. NS4 demands full CPU capacity as k equals 12. In contrast, ns-3 is close the 100% CPU usage as k equals 6. This can be due to the reason, that ns-3 uses a different routing strategy in looping networks than NS4. ns-3 calculates routes of packets on demand. That causes higher CPU utilization and leads to time intensive calculation at simulation runtime. In contrast, NS4 calculates these paths beforehand and stores them in the memory, leading to a cubic memory demand ($O(n^3)$) which exceeds 6000 MB as k equals 24 (see figure 10b)). On the other hand, ns-3 seems to have a linear memory usage. It doesn’t require more than 400 MB of RAM, not even as k equals 24. Since not all flow table entries of the automatic population are used in every simulation, operators may choose not to generate all of them in order to save up memory.
Figure 10c) depicts execution time, showing that NS4 outpaces ns-3 by significant factor. Whereas NS4 needs for simulating the fat tree network with k equals 24 a total time of 178 seconds, ns-3 is at hours. This can be ascribed to the different, proactive routing strategy used by NS4, which eliminates massive overhead at runtime. Execution time of ns-3 is omitted in figure 10c), since it wouldn’t fit in the graph without leveling performance of NS4 to a flat line along the x axis. Making use of a logarithmic scale wouldn’t help either.



a) CPU utilization (%) b) Memory usage (GB) c) Execution time (seconds)

Figure 10: Performance evaluation of NS4 and comparison with ns-3. Source: [BBK+18]

6 Conclusion

In this paper, NS4 has been presented. It has been shown that it is capable of simulating large-scale P4 enabled networks while maintaining downward compatibility. Compared to its predecessor, conducting simulation has been simplified, code re-writing is no longer necessary and the simulation speed has been improved by significant factor. Operators can now validate correctness of P4 programs and evaluate their performance, which will eventually speed up their development process by far. In my opinion, a new issue comes up with the cubic memory usage. Besides that, the time it takes to calculate automatic population has not been stated when evaluating the simulator's performance. I assume that some considerable overhead has been concealed.

7 Criticism

In this final section, I will level some criticism at the installation of NS4 and at the performance evaluation provided by [BBK+18].

7.1 Installation

Since NS4 is an open source software project and hosted on github [Kua], one can download the necessary files to install NS4 on the own platform. However, NS4 targets only Ubuntu 16.04 as operating system. In contrast, developers of ns-3 provide tutorials for installation on various operating systems [nP18].
The installation of NS4 and of all its dependencies on a newly set up Ubuntu requires a total of 9,062,368 KiB disk space and takes more than 70 minutes. These are, related to other tools, rather big values.
After installation of all packages, the source code needs to be compiled. The process of compilation takes another 20 minutes resulting in an error. The code provided by the developers cannot be compiled. The code was downloaded on July 12th 2018.

7.2 Performance evaluation

The performance of a large scale simulation with NS4 is presented in subsection 5.3.2. For comparison, the same network simulation is conducted with ns-3. As seen in figure 10, NS4 outperforms its predecessor ns-3 by numerous factor. Authors of [BBK+18] state, that this speedup comes from the use of automatic population, i.e. pre-compiled routing information that is stored in the system's memory. They do not provide any evaluation of NS4 performance without automatic population. I suppose that NS4 would not outperform ns-3 by nearly the same order of magnitude, implicating that the presented results are rather a benchmark for the caching algorithm than an evaluation of the actual simulator's performance.

References

[AAB$^+$16] Samar Abdi, Umair Aftab, Gordon Bailey, Bochra Boughzala, Faras Dewal, Shafigh Parsazad, and Eric Tremblay. Pfpsim: A programmable forwarding plane simulator. In *Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems*, ANCS '16, pages 55–60, New York, NY, USA, 2016. ACM.

[BAAZ10] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. *SIGCOMM Comput. Commun. Rev.*, 40(1):92–99, January 2010.

[Bah17] Victor Bahl. Microsoft azure and microsoft research take giant step towards eliminating network downtime, October 2017. Available at: `https://www.microsoft.com/en-us/research/blog/eliminating-network-downtime/`. [Accessed May 31, 2018].

[Bas] Antonin Bas. P4 runtime - a control plane framework and tools for the p4 programming language. Available at: `https://github.com/p4lang/PI`. [Accessed May 31, 2018].

[BBK$^+$18] Jiasong Bai, Jun Bi, Peng Kuang, Chengze Fan, Yu Zhou, and Cheng Zhang. Ns4: Enabling programmable data plane simulation. In *Proceedings of the Symposium on SDN Research*, SOSR '18, pages 1–7, New York, NY, USA, 2018. ACM.

[BDG$^+$14] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[com18] Tcl community. Tcl developer site, 2018. Available at: `https://www.tcl.tk/`. [Accessed May 31, 2018].

[Con18] The P4 Language Consortium. P4.org website, May 2018. Available at: `https://p4.org/`. [Accessed June 21, 2018].

[FBZ$^+$17] 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*, SIGCOMM Posters and Demos '17, pages 105–107, New York, NY, USA, 2017. ACM.

[Fou17] Free Software Foundation. Gnu general public license, version 2 - gnu-project - free software foundation, October 2017. Available at: `http://www.gnu.org/licenses/gpl-2.0.html`. [Accessed May 31, 2018].

[Gro] Gordon Grodtron. A host-compiled simulation framework for early validation and analysis of packet processing on programmable forwarding plane architectures. Available at: `https://github.com/pfpsim`. [Accessed May 31, 2018].

[HC01] Padmaparna Haldar and Xuan Chen. Ns tutorial 2002, November 2001. Available at: `https://www.isi.edu/nsnam/ns/ns-tutorial/tutorial-02/slides/NS_Fundamentals_1-1.pdf`. [Accessed May 31, 2018].

[HP15] D. B. Hoang and M. Pham. On software-defined networking and the design of sdn controllers. pages 1–3, Sept 2015.

[IKM15] Sungjin Im, Janardhan Kulkarni, and Benjamin Moseley. Temporal fairness of round robin: Competitive analysis for lk-norms of flow time. In *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '15, pages 155–160, New York, NY, USA, 2015. ACM.

[Inc] Riverbed Technology Inc. Project octagon - faq. Available at: `http://phx.corporate-ir.net/External.File?item=UGFyZW50SUQ9MTYxNTYyfENoaWxkSUQ9LTF8VHlwZT0z&t=1`. [Accessed May 31, 2018].

[Ins] Information Sciences Institute. The network simulator - ns-2. Available at: `https://www.isi.edu/nsnam/ns/`. [Accessed May 31, 2018].

[Kes97] S. Keshav. Real 5.0 overview, August 1997. Available at: `http://www.cs.cornell.edu/skeshav/real/overview.html`. [Accessed May 31, 2018].

[KSG14] Karamjeet Kaur, Japinder Singh, and Navtej Singh Ghumman. Mininet as software defined networking testing platform. In *International Conference on Communiction, Computing & Systems*, August 2014.

[Kua] Peng Kuang. A p4-driven network simulator. Available at: `https://github.com/ns-4`. [Accessed July 18, 2018].

[Lab97] Lawrence Berkeley National Laboratory. ns version 1 - lbnl network simulator, October 1997. Available at: `https://ee.lbl.gov/ns/`. [Accessed May 31, 2018].

[Lei85] C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, Oct 1985.

[LZP$^+$17] Hongqiang Harry Liu, Yibo Zhu, Jitu Padhye, Jiaxin Cao, Sri Tallapragada, Nuno P. Lopes, Andrey Rybalchenko, Guohan Lu, and Lihua Yuan. Crystalnet: Faithfully emulating large production networks. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 599–613, New York, NY, USA, 2017. ACM.

[ML01] C. McLean and Swee Leong. The expanding role of simulation in future manufacturing. In *Proceeding of the 2001 Winter Simulation Conference (CatNo.01CH37304)*, volume 2, pages 1478–1486 vol.2, 2001.

[MM16] E. Meriam and N. T. Mediatron. Multiple qos priority based scheduling in cloud computing. In *2016 International Symposium on Signal, Image, Video and Communications (ISIVC)*, pages 276–281, 2016.

[MZK$^+$17] 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 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 15–28, New York, NY, USA, 2017. ACM.

[nP18] ns 3 Project. ns-3 developers, March 2018. Available at: `https://www.nsnam.org`. [Accessed May 31, 2018].

[p4l] p4lang. Behavioral model. Available at: `https://github.com/p4lang/behavioral-model`. [Accessed May 31, 2018].

[PBY$^+$13] Parveen Patel, Deepak Bansal, Lihua Yuan, Ashwin Murthy, Albert Greenberg, David A. Maltz, Randy Kern, Hemant Kumar, Marios Zikos, Hongyu Wu, Changhoon Kim, and Naveen Karri. Ananta: Cloud scale load balancing. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 207–218, New York, NY, USA, 2013. ACM.

[RH10] George F. Riley and Thomas R. Henderson. *Modeling and Tools for Network Simulation*, chapter The ns-3 Network Simulator, pages 15–34. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[RHLFR08] Thomas R Henderson, Mathieu Lacage, and George F Riley. Network simulations with the ns-3 simulator. *SIGCOMM Comput. Commun. Rev.*, 08 2008.

[UBP11] USC/ISI UC Berkeley, LBL and Xerox PARC. The ns manual (formerly ns notes and documentation), November 2011.

[WC14] W. Wang and G. Casale. Evaluating weighted round robin load balancing for cloud web services. In *2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 393–400, 2014.

[WK18] Thomas Williams and Colin Kelley. Gnuplot, an interactive plotting program, 2018. Available at: `http://www.gnuplot.info/docs_5.3/gnuplot.pdf`. [Accessed May 31, 2018].