A Comparative Analysis of Network Overhead Produced By Popular Software-Defined Networking Controllers

Andreea Pocol, Nandan Thakur, Saralin Zassman, and Adam Campbell
University of Waterloo

Abstract

Due to the ever-growing needs of modern networks, software-defined networking (SDN) is becoming increasingly prevalent. SDN offers an advantage over traditional networks by separating the control plane from the data plane and abstracting network complexity away into a single controller, which facilitates communication between network devices and applications. Mininet, a rapid SDN prototyping tool, is a popular choice for teaching, research, testing and development due to its simplicity, portability, rich features, and transferability to hardware. There are numerous choices when it comes to selecting a controller. Previous studies have used Mininet to compare controllers, but experimental evaluations have been lacking. In this paper, we use Mininet to compare the default Stanford reference, OVS, and POX controllers. We generate realistic flows using iPerf and the Distributed Internet Traffic Generator (D-ITG), evaluate controller performance in terms of network overhead, and use multiple interleaved trials (MIT) to obtain fair and repeatable comparisons. Through statistical analysis, we found that POX generally outperformed the other controllers under the iPerf workloads, and OVS and POX both outperformed the default controller under D-ITG workloads. We also discovered variability in results between workload generators, underscoring the effect of the generator itself and the importance of considering this effect when designing SDN experiments. While such consideration is missing from existing work, our findings should inform future studies.

1 Introduction

Traditional network architectures face many challenges in keeping up with modern network demands. Recently, there has been an explosion in the number of network-connected devices in mobile phones, tablets, and the Internet of Things (IoT). Traditional networks require administrators to configure switches, routers, firewalls, and access control lists for each device added to a network [2]. This complexity is too difficult for administrators to handle. By separating the control plane from the data plane, SDN greatly simplifies network management and configuration. In addition to the management of forwarding devices, SDN controllers handle programming, policy enforcement, network configuration, topology management, and link discovery. Because they form the core of SDN networks, it is critical to evaluate and understand SDN controllers.

Therefore, we compare the performance of three SDN controllers: the Default[1], POX[2] remote controller and Open vSwitch (OVS) controller[3]. We selected these three controllers because the Default and OVS controllers come pre-packaged with Mininet, and POX is one of the most popular remote controllers, as well as markedly popular for prototyping [14]. Since Mininet is a rapid prototyping tool, POX is a logical choice for researchers using Mininet. This allows us to study some of the most accessible, frequently used, and therefore relevant, controllers.

In our paper, we compare the performance of controllers in terms of network overhead. This overhead is measured by the number and types of generated OpenFlow packets. We use iPerf and D-ITG traffic generators to generate realistic flows, and MIT [3] to alternate between our controllers of interest.

2 Background

2.1 Controllers

Controllers reside between the network devices and applications, and handle communication between them [8]. They use the OpenFlow protocol to configure network devices, compute optimal paths through the network, and inform switches on how to direct packets [14]. They also manage and configure the switches dynamically to ensure that the packets are forwarded from source to destination [8].

[1] We use Default to refer to the Stanford reference controller that is set as the default when Mininet is installed.

[2] https://noxrepo.github.io/pox-doc/html/

[3] https://www.openvswitch.org/

A host is a computer device in a network. When a host first needs to communicate with another host, there is a flow table 'miss,' because the flow entry does not yet exist in a switch's flow tables [20]. Next, the switch sends the packet up to the controller, and the reply from the controller may be a flow modification instruction instructing the switch to add a new flow entry to its flow tables. Finally, the flow entry is broadcasted to all network switches, so that they can all update their flow tables as well [9].

Default and OVS are are C language-based controllers built into Mininet. OVS is a simple test controller designed to manage Open vSwitch, an open-source virtual switch supported by major hypervisor platforms [17]. Remote controllers like POX are preferred for more advanced experiments. POX is a Python-based version of NOX, the first controller to support OpenFlow. POX is seen as a successor to NOX due to overall improvements in performance [19].

2.2 OpenFlow Protocol

The OpenFlow standard allows controllers to configure and communicate with network switches via flow tables. To test connectivity between a controller and a switch, OpenFlow offers the **OFPTEchoRequest** and **OFPTEchoReply** packets [15]. The OFPTEchoRequest message body contains randomized data to be echoed back to the requester. The requester verifies the response by comparing the reply with the transaction ID header. **OFPTPacketIn** packets are sent by the switch to the controller to transfer control of a TCP / UDP packet to the controller. This is triggered by a table miss. **OFPTPacketOut** packets are sent by the controller to the switch. They instruct the switch to send a TCP / UDP packet out of a specified port. These OpenFlow packets contains the TCP / UDP packet, or a pointer to it. The message also contains the actions to be performed on the packet. **OFPTFlowMod** packets are sent by the controller to the switch to update flow entries and action buckets. An action bucket is a set of actions to apply to a packet. **OFPTBarrierRequest** packets are used by the controller to control message order and to receive notification upon processing completion. Once a barrier request arrives at a switch, the switch handles all packets arriving before the request. Upon completion of the request, the switch will send a **OFPTBarrierReply** to the controller.

2.3 Mininet

Mininet is used for network application prototyping [14], research, education and trial deployment purposes [8]. Combined with controllers, it assists networks researchers and developers with implementing and testing systems [9]. While other emulation tools exist, Mininet is the only emulator with a Python API that allows users to build custom topologies using Python [14]. It is also easy to use, open-source, and offers a rich variety of features that make it a favourable tool in SDN research [14].

Mininet's ability to emulate reality was evaluated by Hasan *et al.* [9], who found that the values observed in Mininet closely matched expectations in emulating reality.

2.4 Workloads

A workload is a load or amount of work used to measure the performance of a computer system. We measure the network overhead of SDN controllers under multiple workloads, in case some controllers are optimized for certain workloads. In heavier workloads, a controller must handle a greater number of requests per second, and various resource bottlenecks (e.g., link bandwidth) are more likely to occur. In the following subsections, we discuss our choice of two workload generators.

2.4.1 iPerf

iPerf[4] is a tool which uses a client-server model and generates traffic in the form of data streams between server and client nodes [6].

Mininet Flow Generator[5] generates random flows between hosts using iPerf. For each flow, one host from the network is randomly selected as the server, and another is randomly selected as the client. This workload generator exercises the controllers, since they must detect what is happening in the network and react dynamically. The amount of work that the controller does is dictated by the flows. Mininet Flow Generator accepts configurable parameters, including the number of light flows and the number of heavy flows. It also offers parameter recommendations based on CISCO guidelines. We keep these parameters constant at the recommended 45 light flows and 5 heavy flows per 180-second run, so that each controller is subject to the same types and quantities of flows.

2.4.2 D-ITG

D-ITG [5] is an open-source traffic generator that produces network traffic at packet level using customizable properties such as duration, start delay, and number of packets [16]. D-ITG offers multiple different traffic patterns such as uniform, exponential, normal, and Poisson [16]. Like iPerf, D-ITG supports the client-server model.

3 Related Work

Several works have endeavoured to compare the performance of SDN controllers in Mininet.

[4] https://github.com/esnet/iPerf

[5] https://github.com/stainleebakhla/mininet-flow-generator

Noman and Jasim [14] used D-ITG, as well as iPerf, to generate workloads and measure the performance of the POX controller in terms of bandwidth utilization, CPU load, packet loss, packet send rate, and bit rate. George and Jose [8] compared the performances of two controllers, Beacon and Floodlight, in terms of scalability and throughput across various network sizes and topologies. They found that the Floodlight controller was more scalable. Fancy and Pushpalatha [7] compared the POX and Floodlight controllers using various topologies and found that Floodlight outperformed POX in terms of throughput and delay. Arahunashi *et al.* [4] compared the performance of the Ryu, OpenDayLight, Floodlight and ONOS controllers in terms of delay and throughput, across various topologies. They found that Floodlight outperformed the other controllers. Li *et al.* [12] compared the performance of Floodlight and Ryu containers in terms of bandwidth and delay in various network topologies. They found that Floodlight generally requires more bandwidth than Ryu and also generates less latency. Zhu *et al.* [21] compared benchmarking tools and used them to conduct a quantitative comparison of nine controllers. They could not find a clear winner, but rather posited that the best controller for a network depends on the network itself. Tootoonchian *et al.* [18] used Cbench to compare the performance of four SDN controllers: NOX, NOX-MT, Beacon, and Maestro. They measured controller performance in terms of flow setup throughput and latency because the flow setup process is most often the performance bottleneck [18]. Mamushiane *et al.* [13] used Cbench to study the effect of network load on controller performance. They measured the throughput and latency of ONOS, Ryu, Floodlight and OpenDayLight under various workloads. Jawaharan *et al.* [10] evaluated the performance of three SDN controllers: ONOS, OpenMUL, and POX, in a linear network topology using Mininet and Cbench. Latah and Tocker [11] measured network throughput under gradually increasing workloads using POX, Ryu, Floodlight, ODL, and ONOS.

We found that the experiments above lacked a meaningful evaluation setup, and metrics used for evaluation did not always not correlate with controller performance. Several authors measured metrics like throughput but made claims about controller performance, even though the former mostly depends on the configuration of the links themselves, not anything directly handled by the controller. If controllers impact link throughput, then an inadequate explanation was given for this relationship. We expect the relationship to be a weak correlation if any exists at all, because controllers simply do not affect link bandwidth and the resulting network throughput. We explain how controllers impact the network in Section 2.1. To the best of our knowledge, there has been no prior work measuring the number and type of packets produced by different controllers, even though this is perhaps the most direct profile of a controller's behaviour and most accurate measure of its overhead. Moreover, previous experiments conducted limited - if any - statistical analysis and used few trials. We not only use a more sizable number of trials, but we also employ the MIT [3] approach and conduct a statistical analysis of our results.

4 Methodology

4.1 Experimental Setup

We conducted our experiments on a VirtualBox v6.1.24 machine. The first experiment used Mininet Flow Generator, which in turn uses iPerf. The second experiment used D-ITG.

Our first, iPerf, experiment was conducted on a MacBook Pro (Retina, 15-inch, Mid 2014) running MacOS Big Sur version 11.2.3, with a 2.5 GHz Quad-Core Intel Core i7 processor and 16 GB 1600 MHz DDR3 memory. The second, D-ITG, experiment was conducted on a MacBook Pro (16-inch, 2019) running MacOS Big Sur version 11.2.3, with a 2.6 GHz 6-Core Intel Core i7 processor and 16 GB 2667 MHz DDR4 memory.

In the virtual machine, we installed Mininet 2.3.0, Mininet Flow Generator[6], tshark v2.6.10, and iPerf v2.0.10, and POX[7]. The OVS controller is already installed in `/usr/bin/ovs-testcontroller`. The default controller is pre-installed in Mininet `/usr/local/bin/controller`.

4.2 Experiments

We conducted 15 interleaved trials, measured the numbers of common packet types, and reported the means and standard deviations, to compare the performance of each controller. We also perform a two-sample t-test with 95% confidence to confirm our statistical analysis[8].

In our first experiment, we used Mininet Flow Generator. This generator passes bandwidth arguments to iPerf. Light flows range from 100 KBps to 100 MBps, while heavy flows range from 10 MBps to 10 GBps. These bandwidth arguments will test the network under different volumes of traffic. This test simulates a network application that requires both small and large data flows between hosts. Stressing the network by generating traffic that exceeds Mininet's maximum bandwidth of 1,000 Mbps[9] introduces network congestion and increases network latency.

In the second experiment, we modified Mininet Flow Generator to use D-ITG rather than iPerf. This D-ITG workload generator uses a flow rate of a thousand 512-byte packets per second following an exponential distribution.

[6] https://github.com/stainleebakhla/mininet-flow-generator/

[7] https://github.com/noxrepo/pox

[8] https://anonymous.4open.science/r/mininet-sdn-controllers-perf-1614/supplementary_material_experiment_results.pdf

[9] https://github.com/mininet/mininet/blob/master/mininet/link.py#L246

The D-ITG and iPerf workloads we generated are used to simulate different network applications. The D-ITG workloads we generated did not exceed the network bandwidth in the same way iPerf did. Rather, these workloads generated identical, small streams, in order to simulate a more stable network application. We thus emulate networks with consistent data transfer as well as highly variable networks. We employ these varied workloads to study controller performance under varied network behaviours; the different profiles of these workloads bring out different controller actions. Network congestion could lead to the controller suggesting alternate routes, and thus sending more FlowMod and PacketOut messages. When the network is not as congested, such as under the D-ITG workloads, the network will approach a steady state, and continuous updates to flow tables may not be required.

We use tshark to capture packets throughout the duration of each trial. We created a Python script called `openflow_packet_parser.py` to parse the captured packets [10], as well as Bash scripts [11] [12] for running our experiments, both of which we made publicly accessible.

The following is a pseudocode representation of the experiment scripts. It was run once with iPerf and once with D-ITG. The difference between these experiments comes from modifying Mininet Flow Generator to use D-ITG as the traffic generation mechanism instead of iPerf.

Algorithm 1 Workflow Generation Pseudocode

```
controllers ← {POX, Default, OVS}
numTrials ← 15
for numTrials do
  for all controller in controllers do
    - Set up the controller, starting it as a background
    process if necessary
    - Clean up any existing Mininet topologies
    - Run Mininet Flow Generator, passing in parameter
    values using expect
    - At the same time, launch tshark to capture packets
    for the duration of the workload
    - Kill the controller, if necessary
    - Run openflow_packet_parser.py on the captured packets
    - Results from all runs are stored in a CSV file
  end for
end for
```

We use MIT to obtain fair and repeatable comparisons in a virtual environment. With MIT, a single round consists of running each alternative once. Figure 1 shows the network topology used across all experiments, consisting of five hosts, five OpenFlow switches, and a single controller.

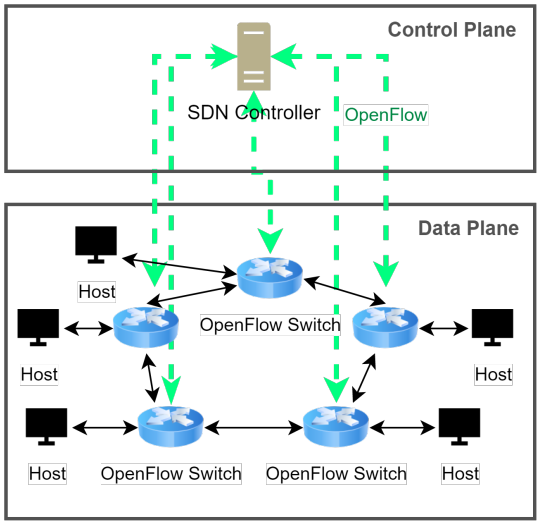

Figure 1: Linear network topology

4.3 Issues and Considerations

We observed that, by default, different OpenFlow versions were used by different controllers. For example, the POX controller used OpenFlow v1 whereas the OVS controller used OpenFlow v4 by default. Python's `scapy` library could not parse OpenFlow v4 packets. Furthermore, for fairness, we needed to ensure that all controllers used the same standard version of OpenFlow. Mininet should theoretically propagate OpenFlow protocol version to the switches in the network. However, when attempted via the Python API, packets still used the old OpenFlow version. Therefore, we hard-coded the OpenFlow protocol in the OVS controller's constructor (`mininet/mininet/node.py`) and rebuilt Mininet.

We needed to ensure that there was no significant difference in default configuration between the controllers which could account for differences in performance. The POX, OVS, and Default controllers all use layer-2 MAC learning for topology discovery in the network [1, 15]. By default, each controller uses exact match flows, meaning that more flows need to be installed, since no wildcards are used in flow matching. In each controller, flows will remain in switches for 60 seconds after their last use.

5 Experimental Results

At the outset of this study, we did not know – due to a lack of existing studies backed by statistical tests – whether there would even be a significant difference between controllers.

[10] https://anonymous.4open.science/r/mininet-sdn-controllers-perf-1614/openflow_packet_parser.py

[11] https://anonymous.4open.science/r/mininet-sdn-controllers-perf-1614/measure-controller-overhead.sh

[12] https://anonymous.4open.science/r/ditg-mininet-controller-overhead-DA8E/script.sh

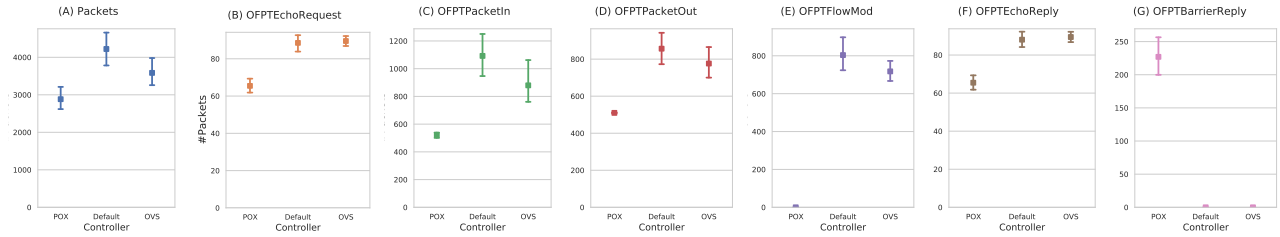

Figure 2: Controller Packet Analysis using **iPerf** for 15 MITs with a 95% confidence interval. The POX controller significantly outperforms both Default and OVS controllers.

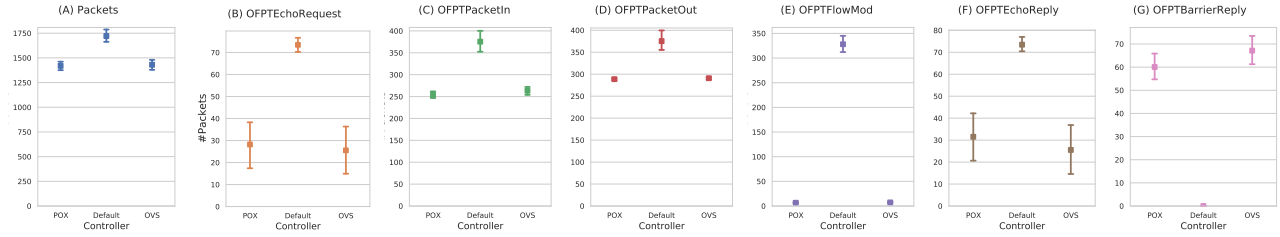

Figure 3: Controller Packet Analysis using **D-ITG** for 15 MITs with a 95% confidence interval. The POX and OVS controllers perform similarly when evaluated with D-ITG.

With iPerf, we observed a statistically significant difference between POX and the other two controllers. This relationship can be seen in Figure 2, which shows our packet analysis with 95% confidence intervals. In the D-ITG experiment, we observed the following (as shown in Figure 3): POX and OVS sent fewer packets than Default for all packet types except BarrierReply. Figure 4 and Figure 5 are boxplots, where each plot shows the average number of packets generated by each SDN controller throughout the experiments, as well as the variability. Overall, we observed a higher variability in the D-ITG experiments, especially in EchoReply and EchoRequest packets, because of the higher variance in flow size.

By examining each packet type, we investigate the overhead and thus 'cost' of each controller. Each packet produced contributes to overall network overhead, and costs bandwidth. Ideally, this overhead should be the minimum required to accomplish the controller's tasks. Overall, POX outperformed the default controller in reducing network overhead for both experiments.

Total Packets: In Figure 3 (A), we observe that, with D-ITG, POX and OVS send fewer total packets than Default. In the iPerf experiment, Figure 2 (A) shows that POX sends fewer total packets than both Default and OVS. This suggests that POX had the lowest overhead overall.

EchoReply and EchoRequest: Using D-ITG, POX and OVS sent a similar number of EchoReply and EchoRequest packets, while Default sent significantly more. In the iPerf experiment, POX sent fewer replies and requests than Default and OVS. POX uses less packets than Default for both experiments, and has the lowest overhead overall for this packet type.

PacketIn: In the D-ITG experiment, PacketIn messages were consistently sent at higher volumes with Default than with POX and OVS. With iPerf, POX sent fewer PacketIn messages than Default and OVS. POX therefore uses less packets than Default for both experiments, and has the lowest overhead overall for this packet type.

PacketOut: POX outperforms Default with regards to this packet type. For the D-ITG experiment, POX and OVS sent fewer PacketOut messages than Default. When iPerf was used, POX sent fewer PacketOut messages than OVS and Default.

BarrierReply: In both experiments, Default sent no BarrierReply messages. With iPerf, POX sent the most such packets, and POX and OVS sent more than Default in the D-ITG experiment. It would seem that POX generated the most overhead for this packet type. That said, this packet type helps organize communication into batches, and since POX seems to require fewer packets of other types, it seems plausible that the barrier packets are actually helpful in reducing overall overhead.

FlowMod: In the D-ITG experiment, Default sent significantly more FlowMod packages than any other controller. POX sent very few FlowMod messages overall, and Default sent many. Recall that these packets change the flow tables, resulting in network and computational overhead, in addition to the standard bandwidth cost all packets incur. Thus, POX outperforms the default controller greatly in minimizing the number of messages of this type sent over the network.

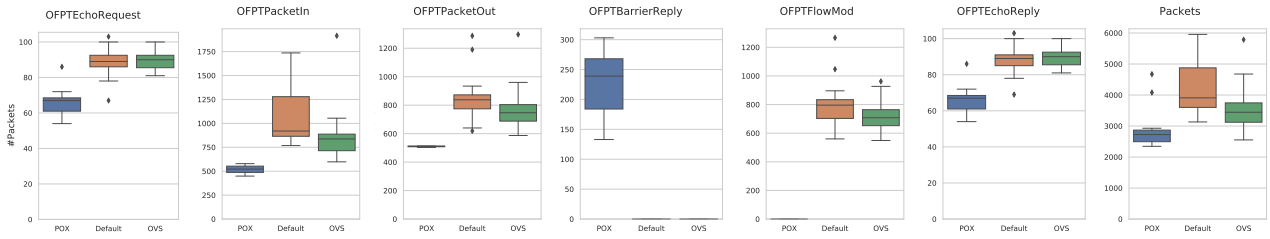

Figure 4: Boxplots using **iPerf** for 15 multiple interleaved trials.

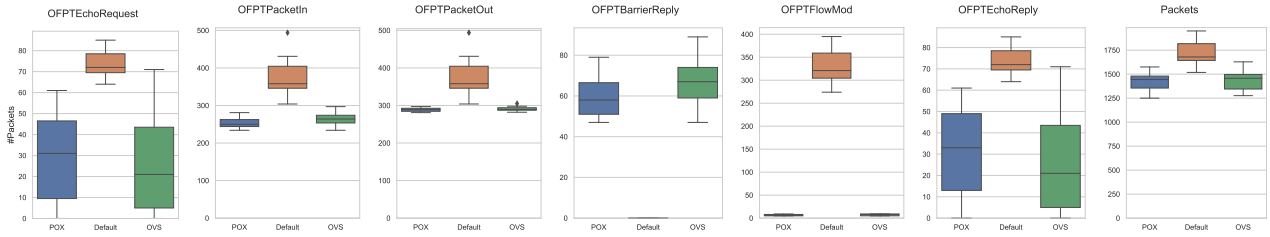

Figure 5: Boxplots using **D-ITG** for 15 multiple interleaved trials.

Overall, with the exception of the BarrierReply messages, POX uses significantly fewer packets to perform the same tasks as the default controller. The OVS controller was most affected by the change in workload, as seen in Figure 2 and Figure 3. This suggests that its performance is the least stable under changing network conditions. The OVS controller performed better relative to the other controllers under the D-ITG workload than the iPerf workload. Our results thereby demonstrate the importance of using multiple workloads when evaluating controller performance. Some controllers may be designed to work better under certain workloads.

Overall, the controllers send less packets under the D-ITG workloads than under the iPerf workloads. Recall from Section 4.2 that the D-ITG workloads we generated did not produce the same flows as the iPerf workloads; iPerf tests the controllers with heavier flows. In Section 4.2, we mentioned that there would likely be more network congestion in the first, iPerf, experiment, leading the controllers to suggest alternate routes, and thus resulting in more FlowMod and PacketOut messages. Our findings seem consistent with this intuition.

The network approaches a steady state with D-ITG and no controller clearly demonstrates the best performance. Only when the network is congested, as in the iPerf experiment, do we see the POX controller outperform both the OVS and Default controllers. This showcases the POX controller's ability to handle heavy network loads with minimal overhead.

Note that we have made our data publicly available [13].

[13] https://anonymous.4open.science/r/mininet-sdn-controllers-perf-1614/supplementary_material_experiment_results.pdf

6 Conclusions and Future Work

Many researchers have sought to compare SDN controller performance, but their findings were not necessarily meaningful or supported by statistical analysis. We wished to investigate whether there truly is any significant difference between controllers, or whether researchers should stop studying this problem. We also considered the ways in which controllers could reasonably differ in the first place. The most direct profile of a controller's unique behaviour and approach is the OpenFlow packets produced while the controller is running in the network, so this is what we studied. In this work, we studied the number and types of packets produced using both iPerf and D-ITG in Mininet Flow Generator, in order to reason about the network overhead introduced by each controller. We observed that the controllers sent more packets under the D-ITG workload than the iPerf workload. This underscores the importance of using multiple workloads when evaluating controller performance. Some controllers may be designed to work better under certain workloads, and workloads themselves are inherently different. POX was found to outperform the other controllers overall, but this was not the only noteworthy result. We further concluded that, when comparing controller performance, it is important to run meaningful tests and reason about how controllers themselves actually impact the network – as opposed to other factors like link bandwidth – run multiple workloads, and use statistical tests to verify the significance of observed results, or lack thereof. In the future, we aim to expand our experiments to include more controllers, topologies, workload generators, and trials, in order to improve confidence in, and applicability of, our results.

References

[1] Ubuntu manpage: ovs-controller - simple openflow controller reference implementation. http://manpages.ubuntu.com/manpages/trusty/man8/ovs-controller.8.html.

[2] Software-defined networking: the new norm for networks. In *Open Networking Foundation*, 2012. http://opennetworking.wpengine.com/wp-content/uploads/2011/09/wp-sdn-newnorm.pdf.

[3] Ali Abedi, Andrew Heard, and Tim Brecht. Conducting repeatable experiments and fair comparisons using 802.11n mimo networks. *ACM SIGOPS Oper. Syst. Rev.*, 49:41–50, 2015. https://dl.acm.org/doi/10.1145/2723872.2723879.

[4] Arun K. Arahunashi, S. Neethu, and RAVISH ARADHYA H V. Performance analysis of various SDN controllers in Mininet emulator. *2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, pages 752–756, 2019. https://ieeexplore.ieee.org/document/9016693.

[5] Alessio Botta, Alberto Dainotti, and Antonio Pescapè. A tool for the generation of realistic network workload for emerging networking scenarios. *Computer Networks*, 56(15):3531–3547, 2012. http://dx.doi.org/10.1016/j.comnet.2012.02.019.

[6] Jon Dugan, Seth Elliott, Bruce A. Mah, Jeff Poskanzer, and Kaustubh Prabhu. iPerf - The ultimate speed test tool for TCP, UDP and SCTP. https://iperf.fr/.

[7] C. Fancy and M. Pushpalatha. Performance evaluation of SDN controllers POX and Floodlight in Mininet emulation environment. In *2017 International Conference on Intelligent Sustainable Systems (ICISS)*, pages 695–699, 2017. https://ieeexplore.ieee.org/document/8389262.

[8] Maria George and Deepa V. Jose. Comparative analysis of performance of controllers in software defined networks using Mininet. 8, July 2019. https://tinyurl.com/235zc9p9.

[9] Muhamad Hasan, Hisham Dahshan, Essam Abdelwanees, and Aly Elmoghazy. SDN mininet emulator benchmarking and result analysis. In *2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES)*, pages 355–360, 2020. https://ieeexplore.ieee.org/document/9257913.

[10] Ragaharini Jawaharan, Purnima Murali Mohan, Tamal Das, and Mohan Gurusamy. Empirical evaluation of SDN controllers using Mininet/Wireshark and comparison with Cbench. In *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–2, 2018. https://ieeexplore.ieee.org/document/8487382.

[11] Majd Latah and Levent Toker. Load and stress testing for SDN's northbound API. *SN Applied Sciences*, 2(1):1–8, 2020. https://link.springer.com/article/10.1007/s42452-019-1917-y.

[12] Yanzhen Li, Xiaobo Guo, Xue Pang, Bo Peng, Xiaoyue Li, and Peiying Zhang. Performance analysis of Floodlight and Ryu SDN controllers under Mininet simulator. In *2020 IEEE/CIC International Conference on Communications in China (ICCC Workshops)*, pages 85–90, 2020. https://ieeexplore.ieee.org/document/9209935.

[13] Lusani Mamushiane, Albert Lysko, and Sabelo Dlamini. A comparative evaluation of the performance of popular SDN controllers. In *2018 Wireless Days (WD)*, pages 54–59, 2018. https://ieeexplore.ieee.org/document/8361694.

[14] Haeeder Munther Noman and Mahdi Nsaif Jasim. POX controller and open flow performance evaluation in software defined networks (SDN) using mininet emulator. 881:012102, August 2020. https://iopscience.iop.org/article/10.1088/1757-899X/881/1/012102/meta.

[15] ONF. OpenFlow switch specification. Technical report, Open Networking Foundation, October 2013. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf.

[16] Georgios Z. Papadopoulos. *Experimental Assessment of Traffic Generators*. PhD thesis, 07 2012. http://georgiospapadopoulos.com/docs/Thesis/MSc-Thesis.pdf.

[17] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. The design and implementation of open vswitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 117–130, Oakland, CA, May 2015. USENIX Association. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff.

[18] Amin Tootoonchian, Sergey Gorbunov, Yashar Ganjali, Martin Casado, and Rob Sherwood. On controller performance in software-defined networks. volume 54, pages 10–10, 04 2012. https://

www.usenix.org/system/files/conference/hot-ice12/hotice12-final33_0.pdf.

[19] Ryhan Uddin and Md Fahad Monir. Performance analysis of SDN based firewalls: POX vs. ODL. In *2019 5th International Conference on Advances in Electrical Engineering (ICAEE)*, pages 691–698, 2019. https://ieeexplore.ieee.org/document/8975667.

[20] Adam Zarek. OpenFlow timeouts demystified, 2012. https://security.csl.toronto.edu/papers/zarek_mscthesis.pdf.

[21] Liehuang Zhu, Md M. Karim, Kashif Sharif, Chang Xu, Fan Li, Xiaojiang Du, and Mohsen Guizani. SDN controllers: A comprehensive analysis and performance evaluation study. *ACM Comput. Surv.*, 53(6), December 2020. https://doi.org/10.1145/3421764.