Experimental Evaluation of Two OpenFlow Controllers

Mohamad Darianian
Department of Computer Science
University of Calgary
Calgary, AB, Canada
mohamad.darianian@ucalgary.ca

Carey Williamson
Department of Computer Science
University of Calgary
Calgary, AB, Canada
carey@cpsc.ucalgary.ca

Israat Haque
Department of Computer Science
Dalhousie University
Halifax, NS, Canada
israat.haque@gmail.com

***Abstract*—Software-Defined Networking (SDN) can help simplify the management of today's complex networks and data centers. SDN provides a comprehensive view of the network, offering flexibility and easing automation. In SDN, traffic management functionality requires a high-performance and responsive controller. In this paper, we conduct an experimental evaluation of two open-source distributed OpenFlow controllers, namely ONOS and OpenDaylight. Specifically, we construct a testbed and use a standard benchmarking tool called Cbench to evaluate their performance. We benchmark the throughput, latency, and thread scalability of these two controllers in both physical and virtualized (OpenStack) environments. The experimental results show that ONOS provides higher throughput and lower latency than OpenDaylight, which suffers from performance problems on larger network models. Additional experiments demonstrate the effects of thread placement on the performance of these two controllers.**

I. Introduction

The art of network design and operation has turned into a remarkably complicated task. This is chiefly because of the rapid development of modern technologies, ever-growing network traffic, and increasing demands of dynamic applications from the network. The emergence of Software-Defined Networking (SDN) has revolutionized computer networking. In essence, SDN improved flexibility, fostered innovation, and enhanced scalability in network operation.

The controller is an integral part of software-defined networks. In such networks, the applications and services enforce high-level policies on the underlying network via the controller. Based on those policies, the controller handles each type of traffic differently, by routing the traffic through certain paths, etc. A plethora of controllers have been proposed. The centralized controllers (e.g., NOX, Ryu, Floodlight, etc.) were designed for early stages of SDN. Subsequently, distributed controllers came into place to address the single-point-of-failure issue and limited capacity of centralized controllers. Among all open-source distributed controllers, ONOS (Open Network Operating System) [1] and OpenDaylight [2] are receiving significant attention since they are modular, support numerous features (e.g., multi-vendor support, cluster configuration, high availability), and have a large technical community. ONOS is supported by the Open Networking Foundation (ONF) and Open Networking Lab (ON.Lab). OpenDaylight is a large SDN open-source project supported by the Linux Foundation, and has many industry partners and collaborators.

These two distributed controllers are the primary candidates for deployment in enterprise-scale software-defined networks. To the best of our knowledge, there is no recent study in the literature focusing on the performance evaluation of ONOS and OpenDaylight. ONOS and OpenDaylight have multiple releases and are maturing in their development.

To better understand performance improvements (if any), it is important to evaluate the latest releases of these controllers. As such, the primary motivation of this work is to conduct an experimental evaluation of ONOS (Goldeneye version) and OpenDaylight (Beryllium-SR2 version). The second motivation of this study is to answer the following question: how well do ONOS and OpenDaylight perform in virtualized environments? In today's IT world, enterprises often virtualize their infrastructure. Thus, understanding the performance of these controllers in a virtualized environment is quite timely and useful. Also, in this paper, we answer to a research question regarding memory management and CPU utilization of OpenDaylight.

The remainder of this paper is structured as follows. Section II reviews prior related work on performance evaluation of OpenFlow controllers. Section III describes our methodology and testbed. Section IV presents the baseline results from our experiments. Section V provides supplementary results exploring three performance issues in more detail. Finally, Section VI concludes the paper.

II. Related Work

In the past few years, there have been numerous studies focusing on the performance of OpenFlow controllers. The majority of these presented benchmarking studies of some well-known controllers (e.g., Beacon, Maestro, NOX). These studies were designed to identify the baseline performance of controllers, and determine which controller outperforms the others in certain tests. On the other hand, some other studies proposed new controllers with advantages over other controllers regarding throughput and latency.

Tootoonchian et al. [5] evaluated the performance of several OpenFlow controllers. The authors presented NOX-MT, a multi-threaded version of NOX controller, and designed a benchmarking tool called Cbench. Cbench enables researchers to assess the throughput and latency of OpenFlow controllers by emulating a network topology of OpenFlow switches.

A preliminary study of the OpenDaylight controller is conducted using Cbench in [9]. The performance of OpenDaylight is compared with Floodlight. The authors noticed that OpenDaylight has unexpectedly poor performance in throughput and latency tests compared to Floodlight. They speculated that this could be due to a memory leak in the OpenDaylight controller. In Section V, we show that OpenDaylight does not suffer from a memory leak. Rather, it has excessively high CPU utilization that degrades the performance of OpenDaylight.

Salman et al. [10] studied the performance of well-known centralized and distributed OpenFlow controllers. The results of their experiments indicate that controllers written in C (e.g., MUL and Libfluid_MSG) have the highest throughput. Next best were Java-based controllers, such as Beacon, IRIS, and Maestro, which demonstrated higher throughput compared to other controllers. They mentioned that controllers coded in C and Java could use many threads, whereas Python-based controllers do not show better performance when increasing the number of threads.

III. Methodology and Experiment Setup

Our methodology studies performance and thread scalability. In this work, we quantify the performance of ONOS [1] and OpenDaylight [2] with Cbench [11]. Our experimental evaluation examines the performance of single-node ONOS and OpenDaylight controllers. Understanding the performance of a single-node is useful since it provides a baseline to compare ONOS and OpenDaylight with each other, and to other OpenFlow controllers.

We conducted our experiments on stand-alone ONOS and OpenDaylight in three phases namely physical server (bare-metal) without Hyper-Threading, physical server with HT, and virtual host.

A. Testbed Setup

The testbed consists of three Cisco UCS C240 M4SX rackmounted servers. Each server is equipped with two Intel Xeon E5-2670 CPUs (24 cores @ 2.30 GHz), 256 GB RAM, 4 TB hard drive, and two NICs[1] (1 Gbps and 10 Gbps). All servers run Ubuntu 14.04.4 LTS x64 operating system. The servers connect through a Cisco UCS 6248 Fabric Interconnect that operates in Ethernet Switching mode.

The experiments on the physical host use two servers. One server is used to run Cbench, and the second server is used to run the controllers (one controller at a time). On the other hand, for experiments on the virtual host, we deploy OpenStack Kilo version on three servers. One server has the role of OpenStack controller, and the other two servers are compute nodes.

[1]The 10 Gbps NIC is used for network traffic, and 1 Gbps NIC is used for out-of-band management. We separate the network traffic from management traffic in our testbed.

We run controllers and Cbench in separate VMs on separate compute nodes for consistency with settings on the physical host. We set the overcommit ratio to 1 in the Nova configuration, so that there is a direct one-to-one mapping between virtual CPUs (vCPU) and physical CPUs (pCPU). Similarly, 1 GB RAM of each VM corresponds to 1 GB RAM on the bare-metal host. For the controller VM, we use 16 vCPUs and 16 GB of memory for running the controller. The remaining 4 vCPUs and 4 GB memory are used to run the operating system.

Each server has 24 cores, divided evenly across 2 CPU sockets. Without Hyper-Threading, each processor has 12 threads. Threads 0-11 reside on socket 1, and threads 12-23 reside on socket 2. With Hyper-Threading, each processor socket has 24 threads, and there are 48 threads in total. Threads 0-11 and 24-35 reside on socket 1, while threads 12-23 and 36-47 reside on socket 2.

IV. Baseline Results

A. Throughput Results

Figure 1 and Figure 2 present the average throughput results for ONOS and OpenDaylight, respectively.

Figure 1 shows that the throughput for ONOS scales well up to 16 switches, and then reaches a saturation plateau at about 1.4 million responses/sec. The performance is similar whether Hyper-Threading is on or off. On the virtual host, ONOS's throughput is roughly 13.5% lower than on the physical host. This performance gap indicates the impact of virtualization overhead on ONOS's throughput. With one switch, ONOS delivers 100K responses/sec on the virtual host. As the number of switches is increased, the throughput trends for the physical and virtual hosts are similar, which indicates that ONOS behaves consistently regardless of the underlying infrastructure.

In our experiment, the throughput of ONOS increases until the number of switches reaches the number of threads used to run the controller (16 threads). This trend matches the behavior of other multi-threaded controllers that have been studied in the past. As mentioned in an earlier benchmarking study [8] for multi-threaded controllers, having more connected switches typically leads to better utilization of CPU threads, until the number of switches exceeds the number of available threads.

The performance of OpenDaylight in Figure 2 is dramatically different (note the lower vertical scale on the graph). OpenDaylight's highest throughput is achieved with 8 connected switches, and there is a drastic throughput reduction as the number of switches is increased beyond 8. For OpenDaylight, the throughput with and without HT differ. It has higher throughput on the physical host when Hyper-Threading is enabled. OpenDaylight's throughput on the virtual host is only slightly lower than its throughput on the physical host with HT. Similar to the physical host, this performance degrades beyond 8 switches. This behavior of OpenDaylight has been

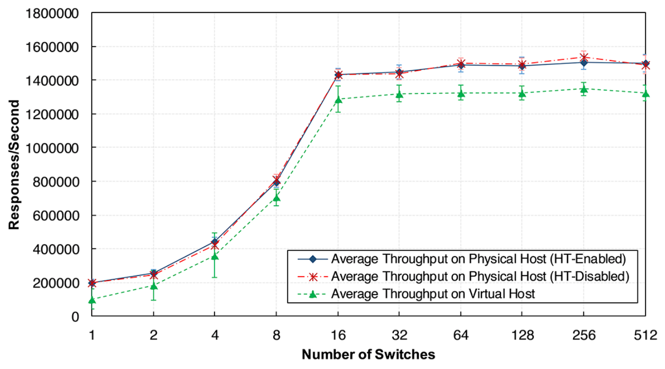


Fig. 1: ONOS throughput vs number of switches

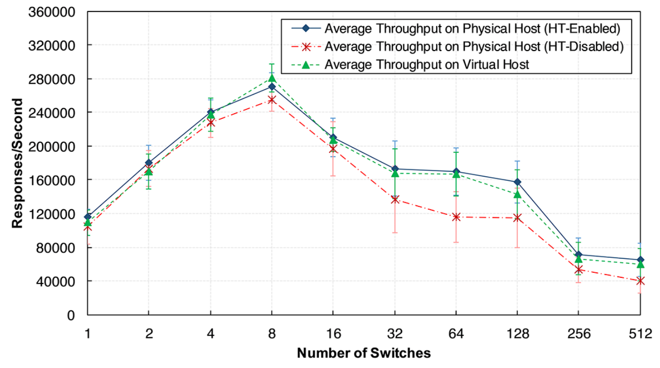


Fig. 2: OpenDaylight throughput vs number of switches

mentioned in the literature [9]. We observed some throughput test failures with 256 and 512 switches. Even though we managed to have some successful tests with 512 switches, OpenDaylight responded with zero in many of our tests. We discarded the failed tests and calculated the average throughput and overall standard deviation based on the successful tests.

B. Latency Results

Figure 3 and Figure 4 depict the average flow setup latency of ONOS and OpenDaylight, respectively. Both controllers demonstrate structurally similar trends in this test. They have better performance with HT enabled on the physical host, whereas the performance degrades on the virtual host. However, the flow setup latency decreases with the increasing number of connected switches.

Note that the latency metric, as directly reported by Cbench, is basically the inverse of the throughput metric. In the latency test, each emulated switch in Cbench sends a *packet_in* message to the controller, and waits to receive the *packet_out* message (response from controller). Cbench measures the delay between the *packet_in* message and the *packet_out* message to calculate flow setup latency of the controller. By increasing the number of switches, the rate of *packet_in* messages in Cbench increases. As a result, the controller receives more *packet_in* messages, which are then processed in parallel. This results in receiving more *packet_out* messages from the controller in the given time, and subsequently decreases flow setup latency. This reflects the Task Batching model implemented in each controller. Task Batching is a method used in multi-threaded controllers to allocate already received *packet_in* messages to the worker threads for processing. Task Batching is one of the key features in multi-threaded controllers that impacts the latency [12].

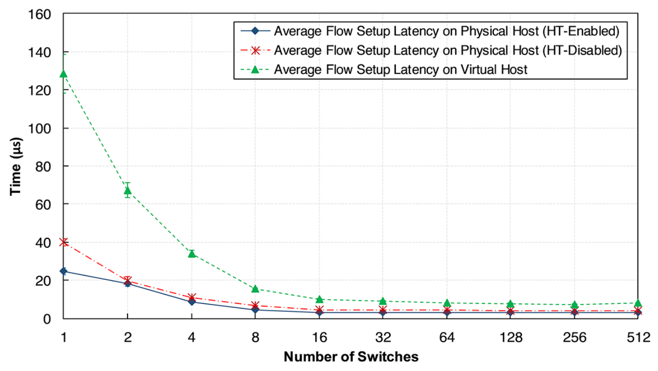


Fig. 3: ONOS flow setup latency vs number of switches

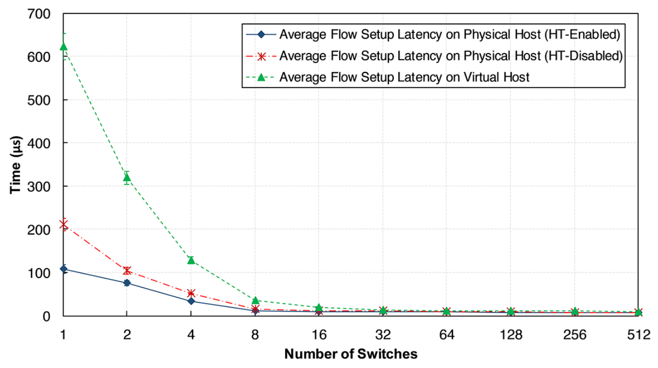


Fig. 4: OpenDaylight flow setup latency vs number of switches

It is worth noting that Cbench only establishes one TCP connection to the controller for all emulated switches. Thus, it gathers aggregated statistics for all switches, but not for each switch individually [13]. As a result, it is not possible to have a finer-grain analysis (e.g., determine per-switch latency or fairness). The observed trend in the latency test, for both ONOS and OpenDaylight, matches the behavior of other multi-threaded controllers that have been studied previously [9] [10] [14].

C. Thread Scalability Results

Figures 5 and 6 illustrate the average throughput of ONOS and OpenDaylight with different numbers of threads. Both controllers exhibit a similar trend in this test, which uses 16 emulated switches. Throughput increases steadily at first, before reaching a plateau for 12-16 threads.

These results illustrate good multi-threading capabilities. Unlike the majority of SDN controllers, ONOS and OpenDaylight are not limited to 8 threads [15], [16], [8], and perform well with up to 16 threads.

ONOS's throughput is similar on the physical host with and without HT. The throughput is slightly better with 6-10 threads on the physical host without HT. Similar to the throughput results with HT, there is a decline in throughput with 12-15 threads. The throughput on the virtual host is lower than that on the physical host, which does not decline with 12-15 threads.

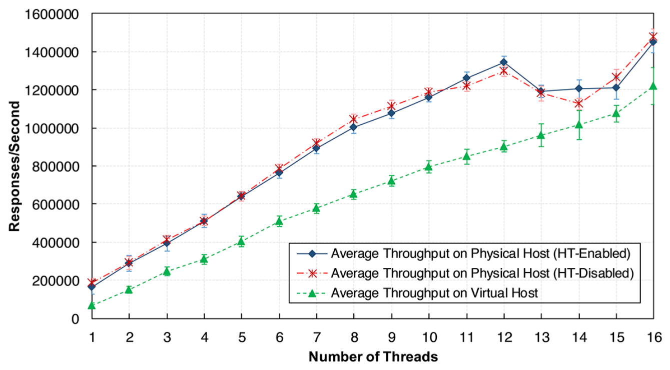


Fig. 5: ONOS throughput vs number of threads

We observed that the ONOS throughput decreased slightly after 12 threads (between 12-15 threads). This trend reflects the communication delay between threads on separate CPU sockets. Recall that the server used to run the controllers has 48 threads (with HT), and 2 CPU sockets. Each processor socket has 24 threads. Threads 0-11 and 24-35 reside on socket 1, and threads 12-23 and 36-47 reside on socket 2. We used the `isolcpus` and `taskset` commands to isolate and bind the desired number of threads to controllers. We isolated and pinned the first 16 threads to each controller at run time. Hence, the first 12 threads (thread 0-11) reside on processor socket 1, and threads 12-15 reside on processor socket 2. Cores (threads) on the same socket typically share L2 and L3 caches, whereas cores on different sockets usually communicate via memory or use a cache coherency protocol forcache-to-cache communications. The additional overhead for this coordination affects the achievable throughput.

The average throughput on the virtual host doesn't show the dip between 12-16 threads, since the CPU allocation to the virtual machine (pinning guest vCPUs to host pCPUs) is automatically handled by the `libvirt` driver in OpenStack Nova. Since the guest operating system (virtual machine) only sees one processor socket, all 16 threads are on the same socket. Hence, further increasing the number of threads could result in better performance on the virtual host compared to the physical machine.

Figure 6 shows the corresponding results for OpenDaylight. The throughput increases with the increasing number of threads. The throughput without HT is lower than that of with HT. The throughput on the virtual host is similar to the results on the physical host with HT.

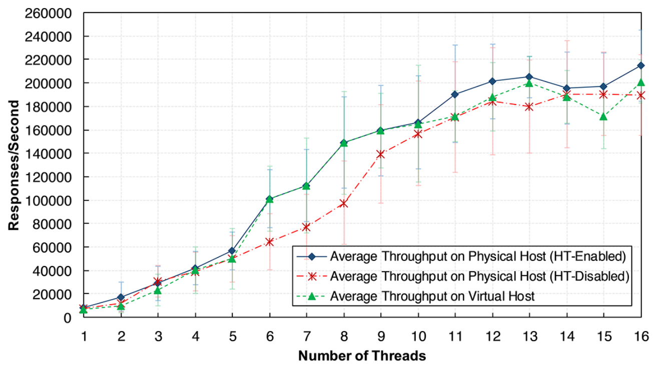


Fig. 6: OpenDaylight throughput vs number of threads

As shown in Figure 6, OpenDaylight's responses in this test show large standard deviations. First of all, it should be pointed out that in all throughput tests, OpenDaylight showed higher standard deviation in its responses compared to ONOS. However, in this experiment, we observed even larger standard deviations, especially when OpenDaylight runs with fewer than 16 threads. We speculate that this may reflect an issue in OpenDaylight's OpenFlow plugin, which fails to process *packet_in* messages in a consistent manner.

Summary: In the throughput test, ONOS dramatically outperforms OpenDaylight. ONOS shows very stable performance, and about 10 times better overall throughput (for 32 switches) than OpenDaylight. The average flow setup latency of ONOS is about half of OpenDaylight's latency. Compared to prior works with earlier versions of these two controllers, ONOS latency has improved by 50%, while OpenDaylight latency is about the same.

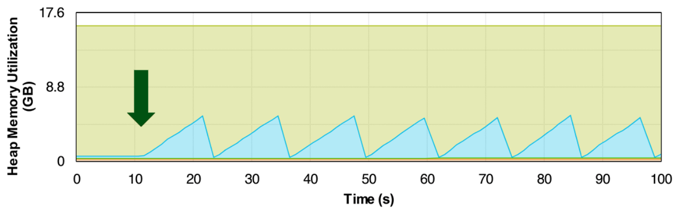
V. Supplementary Results

A. *Memory Usage Results*

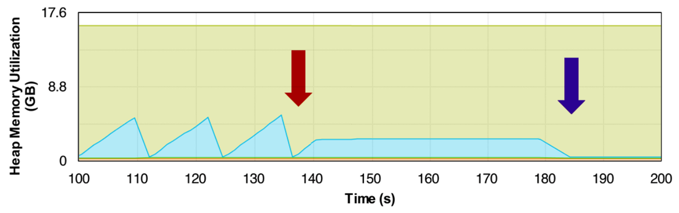
In the discussion section of [9], the authors speculated that OpenDaylight might have a memory leak, which could be the reason for its poor performance in the throughput test. We investigated this issue further, and managed to show that OpenDaylight's memory management is normal, and it doesn't have a memory leak.

To explore this issue, we used YourKit (version 2016.02-b46), one of the most recognized profiling tools for CPU and memory profiling, to scrutinize OpenDaylight memory usage under test load. To capture memory usage statistics of controllers, we used a remote profiling approach.

Figure 7a and Figure 7b present our experimental results on memory usage. Figure 7a presents OpenDaylight's Heap memory usage before and during the throughput test. Figure 7b shows Heap memory usage after test and full garbage collection. We conducted the memory profiling on a fresh install of OpenDaylight, and ran OpenDaylight with 16 threads for this test. We assigned 16 GB of memory to the controller. Following the recommended Java settings for OpenDaylight, we set equal values for the minimum and maximum Heap size (16 GB).



(a) OpenDaylight's memory usage (before and during test)



(b) OpenDaylight's memory usage (after test and full GC)

Fig. 7: OpenDaylight Heap memory usage under throughput test

The throughput test ran for about 125 seconds. In Figure 7a, the green arrow indicates the start of the test. The memory usage growth is normal as OpenDaylight allocates objects for its data. The spikes in the blue area show that the Eden space is almost full. The Eden space usage at most of the spikes is near 5.0 GB. As shown in Figures 7a and 7b, when the Eden space fills, a minor GC occurs. In Figure 7b, the red arrow indicates the end of the throughput test. After that point, the memory is still occupied, which is normal since the Eden space is only partially filled. Hence, no minor GC is triggered.

As shown in Figure 7b, we explicitly forced a full garbage collection (the blue arrow) to reclaim the memory. After full GC, the Eden space and both Survivor spaces became empty. After 10 minor GCs during the throughput test, the usage of old generation is 180 MB, which is very low compared to its limit (10.2 GB). There is no evidence of a memory leak here. In programs with a memory leak, the old generation gets full quite fast, and even with a full GC, the old generation is ever-growing.

B. CPU Utilization

While investigating the memory usage of controllers, we noticed that OpenDaylight's CPU usage was extremely high compared to ONOS. This behavior motivated some additional follow-up tests on both controllers to understand and compare their CPU utilization under throughput test. We used 16 threads to run each controller. Hence, the graphs present total CPU utilization of controllers in each test with 16 threads. Each CPU usage result shows the average from three runs.

ONOS's CPU utilization is less than 10% for one switch, and never exceeds 80%, even for 32 switches. Conversely, OpenDaylight's CPU utilization is about 35% for one switch, and exceeds 96% for 16 switches or more.

As shown in Figure 8, OpenDaylight saturates the CPU. The total CPU usage of OpenDaylight clearly indicates that the main reason behind its poor performance in the throughput test is a CPU-related problem rather than memory leakage.

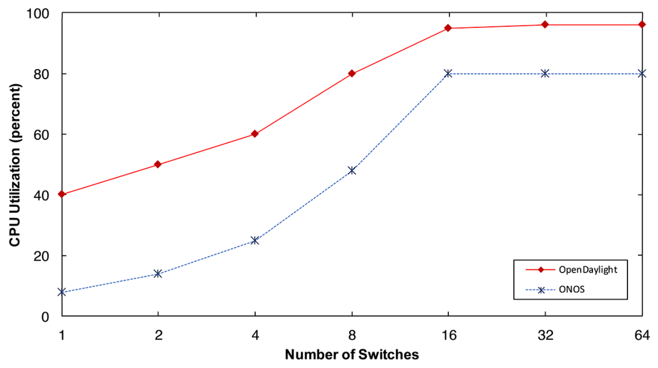


Fig. 8: CPU utilization during throughput test

Considering that ONOS and OpenDaylight are both Java-based controllers (same programming language), and use identical NIO libraries (Netty), they should have similar performance. However, OpenDaylight's surprisingly high CPU utilization, even with one connected switch, suggests a possible bug in OpenDaylight's OpenFlow plugin. Moreover, since the issue is exacerbated by an increase in the number of switches, it is likely that either the Switch Partitioning algorithm or the Packet Batching (*packet-in* scheduling) method are the problem. They do not seem well optimized, which severely affects the controller's CPU utilization and reduces the throughput.

C. Thread Placement

In Section IV-C, we observed a slight decline in throughput with 12-15 threads. We explained the trend and its cause. However, we conducted two follow-up experiments with threads on one processor socket to discover the thread scalability of the controllers without the overhead of inter-socket communication.

Figures 9 and 10 present the average throughput of ONOS and OpenDaylight with different numbers of threads. The yellow line in each graph shows the throughput results with threads on two processor sockets. The purple line illustrates the average throughput with threads on one socket.

On each graph, the throughputs for each configuration are identical up to 12 threads, regardless of whether they are deployed on the same or different sockets. Beyond this point, however, the results diverge. Both controllers show a higher throughput when all threads are on one socket, compared to their performance with threads on two sockets. Also, with threads on one socket, we do not observe the throughput decline in threads 13-15.

Both controllers have similar performance trends in this experiment, except for OpenDaylight's higher standard deviation compared to ONOS. This fact may arise from OpenDaylight's high CPU utilization.

Summary: Our additional experiments have confirmed that the memory utilization of the OpenDaylight controller is normal. The lower throughput of OpenDaylight is attributable

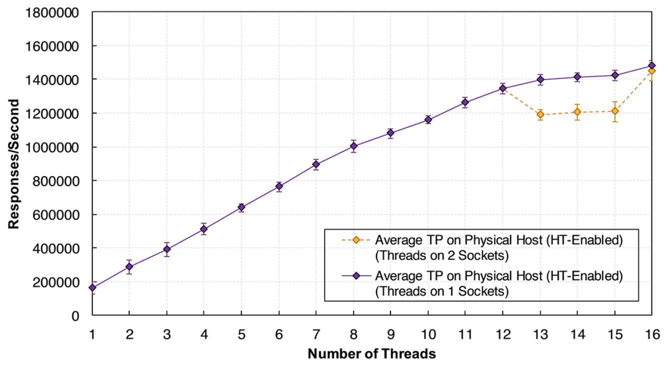


Fig. 9: ONOS throughput vs number of threads (1 socket vs 2 sockets)

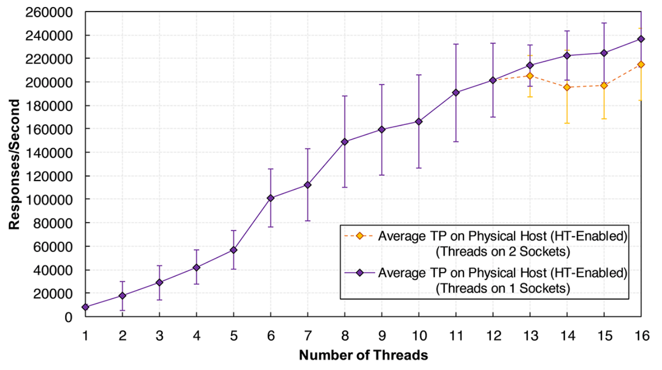


Fig. 10: OpenDaylight throughput vs number of threads (1 socket vs 2 sockets)

to its high CPU utilization. We suspect that this may happen from the Switch Partitioning algorithm or the Packet Batching. As part of our future research, we plan to investigate this issue further. Finally, we observe that better throughput could be achieved by deploying threads on a single CPU socket.

VI. Conclusion

In this study, we showed that the performance of an OpenFlow controller depends on many different factors: controller configuration, underlying network infrastructure, the number of switches, the number of threads, and the placement of threads across sockets. Our experimental evaluation of ONOS and OpenDaylight shows that enabling Hyper-threading results in performance improvement for both controllers. Overall, in most of the tests, ONOS and OpenDaylight show higher throughput and lower latency when Hyper-threading is enabled.

Our study showed that the performance of ONOS and OpenDaylight differ on the virtual host. ONOS's throughput on the virtual host is about 13.5% lower than its throughput on the physical host. On the other hand, OpenDaylight's throughput on the virtual host is similar to its throughput on the physical host. Both controllers had flow setup latency on the virtual host that was two times higher than on the physical host.

Our experimental evaluation of ONOS and OpenDaylight indicates that ONOS outperforms OpenDaylight for throughput and latency. Overall, ONOS shows more scalable and robust performance than OpenDaylight. OpenDaylight's behavior in throughput tests suggests a possible bug in its OpenFlow plugin.

Acknowledgment

Financial support for this research was provided by Canada's Natural Sciences and Engineering Research Council (NSERC), via the Discovery Grant and Strategic Network Grant programs. Additional support was received from MITACS and TELUS. The authors thank Ali Tizghadam, who supported this research and provided access to the TELUS/TRLabs network infrastructure.

References

[1] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an Open, Distributed SDN OS," in *Proceedings of Third ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN 2014)*, Chicago, Illinois, August 2014, pp. 1–6.

[2] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a Model-Driven SDN Controller Architecture," in *Proceedings of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, Boston, Massachusetts, June 2014, pp. 1–6.

[3] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On Controller Performance in Software-Defined Networks," in *Proceedings of the 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, San Jose, California, April 2012.

[4] Z. Khattak, M. Awais, and A. Iqbal, "Performance Evaluation of OpenDaylight SDN Controller," in *Proceedings of the 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, Hsinchu, Taiwan, December 2014, pp. 671–676.

[5] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab, "SDN Controllers: A Comparative Study," in *Proceedings of the 18th IEEE Mediterranean Electrotechnical Conference (MELECON)*, Lemesos, Cyprus, April 2016, pp. 1–6.

[6] "Cbench (OpenFlow Controller Benchmarking Tool)," https://github.com/andi-bigswitch/oflops/tree/master/cbench.

[7] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced Study of SDN/OpenFlow Controllers," in *Proceedings of the 9th ACM Central & Eastern European Software Engineering Conference in Russia*, Moscow, Russia, October 2013, p. 1.

[8] S. Shah, J. Faiz, M. Farooq, A. Shafi, and S. Mehdi, "An Architectural Evaluation of SDN Controllers," in *Proceedings of the IEEE International Conference on Communications (ICC)*, Budapest, Hungary, November 2013, pp. 3504–3508.

[9] M. Jarschel, F. Lehrieder, Z. Magyari, and R. Pries, "A Flexible OpenFlow-Controller Benchmark," in *Proceedings of the 2012 European Workshop on Software Defined Networking (EWSDN)*, Darmstadt, Germany, December 2012, pp. 48–53.

[10] A. Sonba and H. Abdalkreim, *Performance Comparison of the State-of-the-art OpenFlow Controllers*. MSc Thesis, Computer and Electrical Engineering Department, Halmstad University, December 2014.

[11] P. Ivashchenko, A. Shalimov, and R. Smeliansky, "High Performance in-kernel SDN/OpenFlow Controller," in *Proceedings of the USENIX Open Networking Summit*, Santa Clara, California, January 2014.

[12] S. Park, B. Lee, J. Shin, and S. Yang, "A High-Performance IO Engine for SDN Controllers," in *Proceedings of the Third IEEE European Workshop on Software Defined Networks (EWSDN)*, London, UK, September 2014, pp. 121–122.