

Performance Analysis of Software-Defined Networking (SDN)

Alexander Gelberger

*Department of Communication Systems
Engineering, Ben-Gurion University
Beer-Sheva, Israel
gelberge@bgu.ac.il*

Niv Yemini

*Department of Communication Systems
Engineering, Ben-Gurion University
Beer-Sheva, Israel
nivye@bgu.ac.il*

Ran Giladi

*Department of Communication Systems
Engineering, Ben-Gurion University
Beer-Sheva, Israel
ran@bgu.ac.il*

Abstract—Software-Defined Networking (SDN) approaches were introduced as early as the mid-1990s, but just recently became a well-established industry standard. Many network architectures and systems adopted SDN, and vendors are choosing SDN as an alternative to the fixed, predefined, and inflexible protocol stack. SDN offers flexible, dynamic, and programmable functionality of network systems, as well as many other advantages such as centralized control, reduced complexity, better user experience, and a dramatic decrease in network systems and equipment costs. However, SDN characterization and capabilities, as well as workload of the network traffic that the SDN-based systems handle, determine the level of these advantages. Moreover, the enabled flexibility of SDN-based systems comes with a performance penalty. The design and capabilities of the underlying SDN infrastructure influence the performance of common network tasks, compared to a dedicated solution. In this paper we analyze two issues: a) the impact of SDN on raw performance (in terms of throughput and latency) under various workloads, and b) whether there is an inherent performance penalty for a complex, more functional, SDN infrastructure. Our results indicate that SDN does have a performance penalty; however, it is not necessarily related to the complexity level of the underlying SDN infrastructure.

I. INTRODUCTION

Communication networks are growing in size and complexity at an ever-increasing rate, with the conventional infrastructure, network systems, and protocol stack, which hardly provide adequate solutions to the contemporary networking demands. This triggered the emergence of a different approach to network systems architecture, called Software-Defined Networking (SDN) [1]. SDN, has been present for the last 20 years, [2]. Recently, OpenFlow [3] succeeded in establishing itself as an SDN industry standard. OpenFlow validated the SDN approach, and many network architectures, network systems, and data centers adopted SDN and turned it into a mainstream approach in network design. We introduced another SDN technology that is based on IETFs ForCES [4], called ProGFE (Programmable Generic Forwarding-Element) [5]. ProGFE is essentially a superset of OpenFlow in terms of capabilities and provides a richer functionality than OpenFlow, although it is an entirely different technology, and is based on an open IETF standard that is targeted in Forwarding and Control Element

Separation.

SDN offers many advantages, such as centralized and decentralized control of multiple cross-vendor network elements, mainly data plane platforms with a common API abstraction layer for all SDN-enabled equipment. It also reduces the complexity of network configuration and operation that is achieved by automation high level configuration is translated into specific forwarding behavior of network elements. SDN allows easy deployment of new protocols and network-services as a result of high operation abstraction. Increased control granularity in SDN allows a per flow definition with a high granularity policy level. SDN infrastructure can adjust to the specific user application running on it via the control plane, which greatly improves the user experience.

Software Defined Networking, however, has its disadvantages: the added flexibility and functionality require additional overhead on the equipment, and as a result there are performance penalties in terms of processing speed and throughput. This is not to say that the overall performance is necessarily decreasing; many network services and tasks that were executed by the end-nodes or by the control or management layers of the network systems can be executed by the SDN-enabled equipment in a simpler and quicker way, thereby improving the overall performance of the networking tasks.

Despite SDN's continuing growth in popularity, there have been relatively few studies that deal with performance evaluation of SDN architectures. Tootoonchian et al. [6] focused mainly on performance evaluation of the control plane of SDN. Rotsos et al. [7] proposed a tool for evaluating performance of one specific SDN architecture, i.e., several OpenFlow implementations, and measured raw performance of OpenFlow without comparison to other SDN solutions, or to non-SDN network systems. Their work indicated that performance is affected by the number and type of actions applied to the data-frame, as well as the specific implementation of the SDN. Another study of data plane performance evaluation of OpenFlow soft switch was carried out by Bianco et al. [8], using different frame sizes and rules. Some of the results of this study were unintuitive, and show that OpenFlow and a more complicated native (non-

SDN) application of routing outperform a simpler native switching application. The authors suspected that the cause was due to inefficient implementation of the switching application. Jarschel et al. [9] evaluated the forwarding speed of OpenFlow, but focused mainly on the effects the controller has on the forwarding plane. Most of the research described above deals with evaluating a single architecture of SDN, OpenFlow; no comparison to other SDN architectures were conducted, while a comparison with native, non-SDN implementations of the networking tasks wasn't always available.

In this paper we analyze two issues: a) how much, if at all, SDN impacts raw performance (in terms of throughput, latency, etc.), and b) whether the performance of the SDN is a function of its complexity or functionality, i.e., whether there is an inherent performance penalty for a complex (more functional) SDN. We found that SDN does come at the expense of performance; albeit, as outlined above, the functionality it provides improves the overall performance of the network in terms of efficiency, i.e., less equipment and resources are required to perform complex networking tasks and applications. However, we found that complexity and functionality do not necessarily affect performance, which depends more on the SDN model and its implementation. Specifically, we found that ProGFE, which offers more functionality and capabilities than OpenFlow, works more efficiently in terms of latency, throughput, and jitter (packet delay variation) than OpenFlow, and ProGFE also offers better efficiency in terms of networking resources, e.g., it performs OAM tasks directly without the need for additional network elements.

II. OPENFLOW AND PROGFE ARCHITECTURES AND DIFFERENCES

OpenFlow and ProGFE are SDN approaches based on the concept of separating the control and forwarding planes. In the following subsections, architectural highlights and differences of both approaches are presented.

A. OpenFlow

OpenFlow [3] is an open SDN standard that consists of two main entities, the OpenFlow switch, which implements the forwarding plane, and the Controller, which implements the control plane. These two communicate via a secure channel that is implemented over a special protocol, also named OpenFlow.

Each packet that arrives at the OpenFlow switch passes through a series of Flow Tables (Figure 1), where one or more of the packet fields are used as a lookup key. If a match is found, an action is performed on the packet, and the packet may be forwarded to another Flow Table, an egress port, or dropped altogether. The Controller configures the operation of the OpenFlow switch by updating its Flow Tables. A Flow Table entry comprises a Rule and a set of actions and

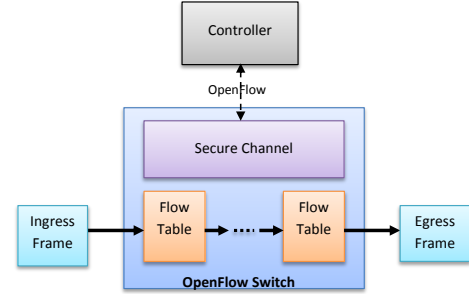


Figure 1. OpenFlow switch.

instructions. A lookup is performed on each arriving frame according to the Rule parameter, and in case of a match the appropriate actions are performed. Each action defines which operation is to be done on the incoming frame, and an instruction defines in what manner to apply the actions (apply now, apply later, etc.).

In the case of an undefined frame that arrives at the switch, it is encapsulated and forwarded to the controller, which handles it according to some more complex logic. This might also trigger an update of the Flow Tables, so that similar frames would not have to go through the controller again. Specific frame types that require more complex decisions, which are not supported by the switch, e.g., OAM, can be configured to always be sent to the controller.

B. ProGFE

Programmable Generic Forwarding Element (ProGFE) [5] is a flexible and reconfigurable network platform, which enables the deployment of various networking services, network protocols, and architectures. This platform is configured on the fly, via an XML-based API. ProGFEs operation is based on IETFs ForCES (Forwarding and Control Element Separation) [4] definition of Logical Functional Blocks (LFBs), where each LFB defines the way the ProGFE operates.

Every processing operation in ProGFE is represented by one or more LFB. These LFBs are connected in the form of a directed graph, and the specific processing of each frame is defined by the path it takes along this graph, depending on the frames type. LFBs are represented in XML format, which makes them easy to read, understand, and define.

The ProGFE contains two groups of modules (Figure 2): management and run-time modules. The management modules interact with the Control Element (CE), receive the LFB information, analyze the information, and create internal data structures that define and control the way the run-time modules process each frame. The three main ProGFE modules are the Protocol Agent and the XML Handler, which are management modules, and the Low Level Rule Enforcer (LLRE), which is a run-time module. The Protocol Agent communicates with the CE using a protocol that governs the communications (e.g., HTTP, Ethernet). The XML Handler parses XML information and translates

the LFB definition to relevant specific data structures that reside in the ProGFE. The LLRE executes a meta-program that is capable of executing pre-defined LFB types, which implement the various LFBs and their parameters, as defined by the CE and received from it. An Adaptation Layer is used for abstracting the hardware implementation of the LLRE to the management modules. Two additional modules are included in the ProGFE to support additional services. Specific incoming frames can be diverted to the ProGFE by the Dispatcher, if programmed to do so by the LLRE. The Dispatcher sends such frames to relevant Service Agents. ProGFE has an existing Network Processor implementation

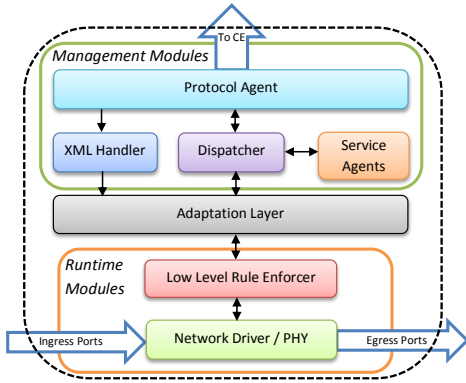


Figure 2. ProGFE block diagram

[5] so that frame processing is achieved at wire speed, even though each frame is analyzed and processed by a meta-program. A general purpose PC version of the ProGFE was also implemented in order to provide a tool to deploy and test new network protocols, on a flexible, inexpensive, easily configurable, and widely available platform. This implementation is based on Linux, which makes the Linux ProGFE hardware-independent, inexpensive, and scalable. Linux ProGFE has an additional layer, the Network Driver, which interacts with the LLRE and is responsible for controlling the PCs NICs, i.e., sending and receiving frames. This layer is part of the LLRE in the NP implementation. Linux ProGFE was written in C++ in user-space, and is available at [10].

C. SDN Differences

ProGFE and OpenFlow, although similar in their general approach of control and forwarding planes separation, differ in the complexity of the forwarding plane. ProGFE has a rich functionality integrated into the forwarding plane, e.g., it can perform learning and searching tasks, stateful operations, and OAM functions, which OpenFlow shifts to the control plane. Even more complex functionality can be added to the ProGFE by additional Service Agents. Another major difference between the two SDNs is that ProGFE provides a higher level of abstraction, i.e., the use of LFBs and defining LFBs by XML, thus ProGFE enables description and implementation of new networking functions and services

in an easy and simple way. Another key advantage is that the ProGFE is indifferent to the structure of the incoming frames, as frame formats are defined by the CE; thus ProGFE is not restricted to static definitions of frame format. This enables ProGFE to support virtually any frame type and any protocol, and execute Deep Packet Inspection (DPI) applications.

III. PERFORMANCE ANALYSIS

Most of the conventional and SDN-based network systems are based on dedicated and powerful platforms. For the purpose of comparing the performance of conventional and SDN-based network systems, and between the two SDN architectures described above, we implemented network systems on a simple PC platform, under the assumption that differences between simple platform performances indicate the differences in performance of dedicated network systems platforms. We verified this assumption by testing the differences in performance using a powerful network-processor-based-platform in addition to the simple PC platform.

A. Methodology

We compared the performance of various workloads (networking tasks) between SDN (OpenFlow and ProGFE), running in the user space of Linux, and non-SDN (dedicated application that executes the networking tasks directly, e.g., pure Linux forwarding) running in the user space (for comparison), and in the kernel space (to explore the PC potential).

We used the Linux ProGFE described in the previous section, and the OpenFlow soft switch v1.0 stable version. The following workloads were examined: IP routing, as a rather simple Forwarding Element task, and VLAN tagging, which represented a more complicated task for an FE.

In order to evaluate performance, we ignored the indirect performance benefits that SDN enables (e.g., reduction in number of network systems, fast implementation of new technologies), and focused on "raw" performance metrics. These metrics include throughput, latency, and jitter. To test the throughput we used a standard tool - iperf to establish a TCP/IP flow with various TCP window sizes, from 5 Kbytes to 10 Mbytes. In order to evaluate the latency, synthetic frames were generated with varying frame sizes, with an additional timestamp that was used to calculate the latency. The Packet Delay Variation (PDV), referred to as jitte, was also examined.

High performance, network processor-based platforms were also used to examine the performance difference between two SDN-based bridge applications (plain and complex bridge implementations) and a dedicated, native bridge application. These platforms were based on EZChips NP-2 Network Processor. Two scenarios were tested: a plain configuration, programmed efficiently as an Ethernet bridge,

and a complex configuration of a bridge, configured with more than 40 different additional workloads.

B. Experimental testbed

The unit under test (UUT), on which the SDNs and the dedicated systems were implemented, was based on an Intel Core2Duo e6600 CPU, running at 2.4 GHz, 2 GB of DDR2 memory, and three Intel 82572EI Gigabit NICs.

For throughput testing we used a simple source and sink testbed architecture, with two common PCs connected via the UUT PC. The two PCs established a TCP/IP flow using iperf, with the UUT performing the switching or routing between them, depending on the defined workload.

A dedicated, embedded MPC8360 microcontroller was used to generate and receive frames to and from the UUT, and to measure the latency and the jitter, in microseconds.

C. Results

In this section we describe the differences in raw performance of the PC-based implementations for the kernel-space and the user-space implementations of the pure Linux forwarding, the user-space Linux ProGFE, and the user-space OpenFlow soft switch, and for the network processor implementation between a basic bridge and the ProGFE.

1) *SDN and native implementations of bridging on network processor*: Figure 3 compares the average latency between a basic, native, Ethernet bridge that was implemented by dedicated software on the network processor (annotated bridge in the figures) and two implementations of NP-based ProGFE bridge, one plain and the other complex (as described above), using NP-2 network processor. The comparison shows that the ProGFE has additional overhead when configured with a more complex functionality, which can reach up to 10 microseconds on average.

It should be emphasized that the throughput was not affected by the bridge implementation, as the NP worked at wire speed in both NP types (1 Gbps was tested). The latency was affected by a factor of about 5 microseconds for the plain implementation of the ProGFE bridging, and an additional 5 microseconds, on average, for the complex implementation of the ProGFE bridging, in both NP types.

2) *Throughput- PC platform*: Figures 4 and 7 compare the throughput between the PC-based implementations of the SDN and the pure Linux forwarding for the three workloads described above (switching, routing, and VLAN). Pure Linux forwarding was measured once as a kernel-space implementation (annotated Linux kernel-space) and once as a user-space implementation (annotated Linux user-space). SDN implementations are annotated as Linux ProGFE and OpenFlow v1.0, and the direct connection between the source and the sink PCs, without the UUT, is annotated Direct connection.

This comparison shows that although ProGFE is the more complex SDN architecture, its performance in terms

of throughput is higher than the less complex OpenFlow SDN architecture, for most frame sizes. In addition, we can observe the distinct advantage in performance that the kernel-space implementation of the Linux forwarding has over the user-space implementations of the tested SDNs.

These results show that complexity of the SDN architectures doesn't necessarily influence the performance of the SDN implementation, but in each implementation, the complexity of the workload does impact performance.

3) *Latency - PC platform*: Figures 5 and 8 show the comparison of results of the processing times (as indicated by the measured latency) in the routing and the VLAN tagging workloads. As shown in Figures 5 and 8, ProGFE has a slightly lower latency for the simple workload of routing, and latency similar to that of OpenFlow for the more complicated workload of VLAN tagging.

4) *Jitter (Packet Delay Variation) PC platform*: Figure 6 shows the jitter results for the routing and VLAN tagging workloads. The user-space ProGFE results are almost as good as a kernel-space Linux, while the results for OpenFlow PDV are quite high for both workloads. In addition, the results show that the complexity of the workload doesn't affect jitter dramatically. The high level of jitter in OpenFlow might explain the fact that although the latency values are nearly the same as in the ProGFE, the throughput is lower. The occasional high delay values result in TCP timeouts, which in turn cause the TCP protocol to reduce its rate, thus decreasing overall throughput.

5) *Discussion*: The scenarios we tested did not include frequent involvement of the controller in both SDNs; despite this the less complex SDN - OpenFlow - did not outperform the ProGFE, and in most tests showed lower performance results. One possible cause for this is an implementation issue of OpenFlow v1.0 that causes very high jitter values, which in turn cause throughput and latency penalties. In scenarios where the Forwarding Element (FE) of a less complex SDN has to consult the controller frequently, the overall performance of that network should be worse, and in some cases completely unusable, since most of the network traffic will consist of FE to CE packets. Additionally, a network that consists of more complex SDN FEs will be much more scalable even for simple scenarios, as a centralized controller can handle a limited amount of control traffic.

IV. SUMMARY AND CONCLUSIONS

In this paper we analyzed the performance of two SDN architectures, OpenFlow and ProGFE, as a function of their complexity, flexibility, and potential functionality and capabilities. We saw that SDN flexibility does come at the expense of raw performance, enhanced by additional overhead for a more complex functionality. This conclusion contradicts the results shown in [8], although the authors suspected that their implementations were not appropriate. We used three different workloads, switching, routing,

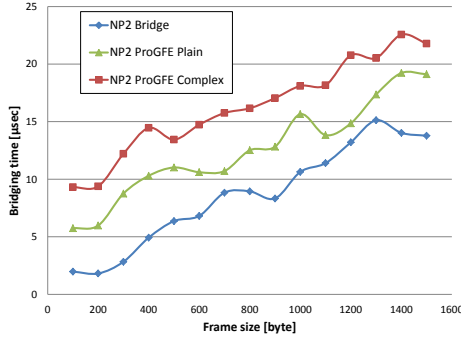


Figure 3. NP-2 ProGFE bridging time.

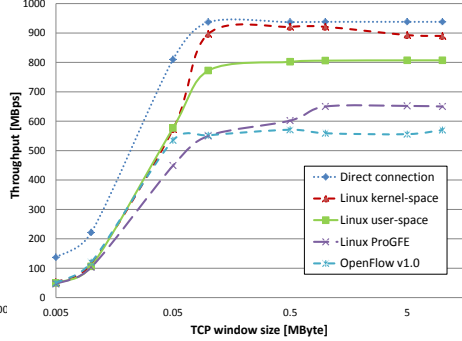


Figure 4. Throughput - Routing.

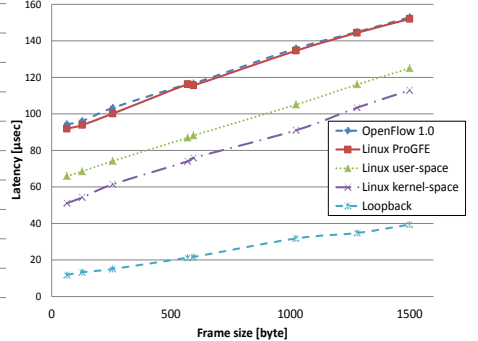


Figure 5. Latency - Routing.

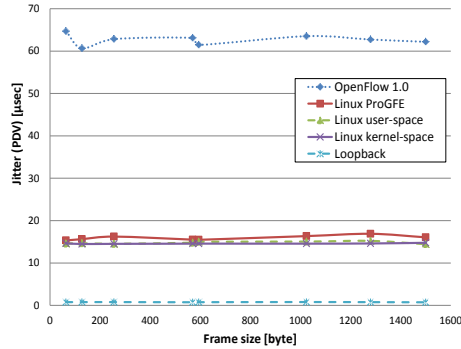


Figure 6. Jitter - Routing.

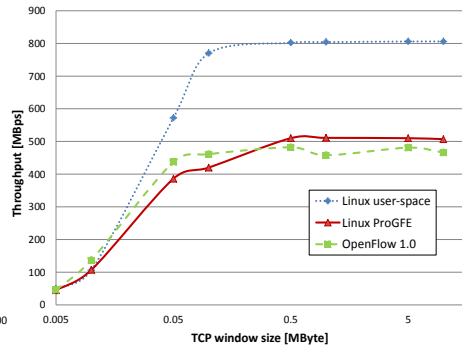


Figure 7. Throughput - VLAN tagging.

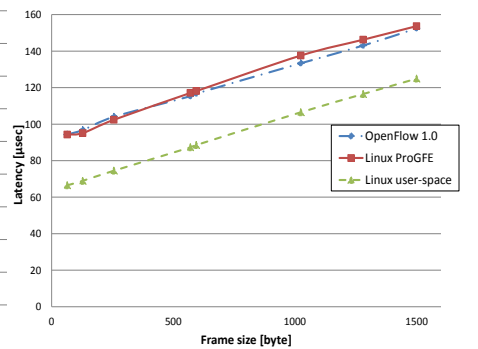


Figure 8. Latency - VLAN tagging.

and VLAN tagging, and saw that for all three workloads ProGFE has better performance than OpenFlow in terms of throughput for most frame sizes, and roughly the same performance in terms of latency. We also saw that the complexity of the workload affects the performance of SDN-based systems in terms of throughput and latency, and this is in agreement with the results of [7]. OpenFlow has a much higher jitter than ProGFE (that is not affected by the workload complexity), which results in lower throughput, despite similar latency.

The results also show that a more complicated SDN, which enables more flexibility, functionality, and capabilities, does not necessarily mean performance degradation. Performance reflects the specific implementation of the SDN. Moreover, in workloads that require the intervention of the control plane in the less complicated SDN, performance differences will be greater, in favor of the more complicated SDN.

REFERENCES

- [1] G. Goth, "Software-Defined Networking Could Shake Up More than Packets," *IEEE Internet Computing*, vol. 15, no. 4, pp. 6–9, 2011.
- [2] D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 26, no. 2, pp. 5–17, April 1996.
- [3] N. McKeown, T. Anderson, H. Balakrishnan, G. M. Parulkar, L. L. Peterson, J. Rexford, S. Shenker, and J. S. Turner, "OpenFlow: enabling innovation in campus networks," *Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.
- [4] L. Yang, R. Dantu, T. Anderson, and R. Gopal, "Forwarding and Control Element Separation (ForCES) framework," RFC 3746, 2004.
- [5] R. Giladi and N. Yemini, "A programmable, generic forwarding element approach for dynamic network functionality," in *Proc. PRESTO*, 2009, pp. 19–24.
- [6] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proc. USENIX Hot-ICE*, 2012.
- [7] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "OFLOPS: An Open Framework for OpenFlow Switch Evaluation," in *Proc. PAM*, 2012, pp. 85–95.
- [8] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "OpenFlow Switching: Data Plane Performance," in *Proc. ICC*, 2010, pp. 1–5.
- [9] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and performance evaluation of an OpenFlow architecture," in *Proc. ITC*, 2011, pp. 1–7.
- [10] "Linux progfe." [Online]. Available: "http://in.bgu.ac.il/engn/cse/Pages/PRO-GFE.aspx"