# P4RCProbe for Evaluating the Performance of P4Runtime-based Controllers

Hasanin Harkous*, Khaled Sherkawi†, Michael Jarschel*, Rastin Pries*, Mu He*, Wolfgang Kellerer†

**Nokia*, firstname.lastname@nokia.com
†*Technical University of Munich*, firstname.lastname@tum.de
Munich, Germany

***Abstract*—The SDN control plane provides the data plane with packet forwarding rules and thus defines its behavior. Understanding the performance of the controller is critical to assess the overall performance of the network. Although OpenFlow-based controllers have been well investigated in this regard, controller performance studies based on P4Runtime, which is the de-facto implementation for the southbound control of programmable data planes in P4, are still missing.**

**In this work, we implement a benchmarking tool for P4Runtime-based controllers, apply the tool to evaluate the performance of the ONOS controller running in both OpenFlow and P4Runtime mode, and identify processing bottlenecks in the P4Runtime implementation. In addition, We propose a code patch for the implementation, which shows a 17 % improvement in the achieved packet rate.**

***Index Terms*—Performance Analysis, Benchmarking Tool, SDN Controller, P4Runtime, OpenFlow**

## I. Introduction

Understanding the performance of Software-Defined Networks (SDN) is key for their development, planning, and deployment. In this regard, the performance of the original SDN approach based on OpenFlow (OF) has been thoroughly investigated over the past decade and more towards the aspect of the gains and costs due to the programmability of the control plane [1], [2]. With the introduction of data plane programmability and the P4 language, the building blocks of a programmable network have changed and thus the performance characteristics deserve additional study. In this regard, we have already investigated the performance of the data plane and the properties of P4 constructs [3]–[7]. However, to fully understand the performance of the complete system, the control channel interaction between the control plane and the data plane also needs to be investigated. Moreover, it is important to understand the possible bottlenecks of the controller's performance or even unexpected behavior under high load scenarios [8]. Currently, the only stable implementation of a P4 controller is ONOS [9], which communicates with the data plane through the gRPC-based P4Runtime API. As ONOS can also function as an OF controller, we can besides investigating the performance of the P4Runtime (P4RT) implementation, also compare it to its predecessor OF using the same software platform and identify performance differences.

For this purpose, we develop a new P4RT benchmarking tool based on OFCProbe, an OF controller benchmarking suite [10]. We perform comprehensive measurements and evaluate the performance in terms of a) control plane processing rate, b) control plane reaction time (processing time plus round-trip time), c) controller Core utilization, and d) outstanding packets (i.e. non-responded packets by the controller). We observe the performance gap in comparison to OF, which mainly stems from the flexibility in terms of the data plane configuration offered by P4. In addition, we identify a potential bottleneck in the implementation of ONOS P4RT and suggest a mitigation strategy, which improves the control plane processing rate by around 17 %.

The paper is organized as follows. Section II clarifies relevant background and prior work. In Section III, the implementation of the newly proposed benchmarking tool P4RCProbe is illustrated. Evaluation results of the ONOS controller in both OF and P4RT modes are elaborated in Section IV. P4RCProbe is used to identify a bottleneck in ONOS P4RT implementation in Section V. Finally, Section VI concludes the work.

## II. Background & Related Work

This section first reviews the P4Runtime API, i.e., the de-facto runtime API for controlling P4 data planes, and the ONOS controller under investigation. Prior art related to benchmarking SDN controllers is then revisited focusing on OFCProbe benchmarking tool, which is the basis for the new P4Runtime benchmarking tool.

### A. P4Runtime Framework

In a nutshell, the P4Runtime [11] is a runtime control API for P4 programmable data planes, which was first released in 2019 and is still actively developed. P4RT is designed to be a) target-independent, i.e., it can be used to control different data planes, b) protocol-independent, i.e., it enables the introduction of new data plane protocols, and c) pipeline-independent, i.e., it allows control of different P4 defined pipelines. The API is based on Google's Protobuf [12] to realize a language- and platform-neutral mechanism for serializing structured data. The endpoints of a P4RT connection reside in the controller and data plane device respectively in the form of client and server implemented with gRPC [13]. gRPC can automatically generate code in different programming languages, provide security mechanisms, and support bi-directional channels for

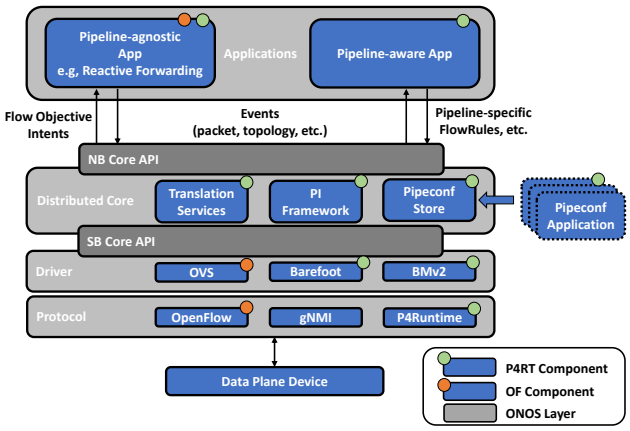



Figure 1: ONOS Architecture, highlighting functional blocks added to support P4RT.

data streams. P4RT also supports pushing new P4 programs to the data plane to reconfigure it during runtime.

When using P4RT, the P4 source code is compiled to generate a P4 device configuration file and a P4Info metadata file. The configuration file is target-specific to program the device, whereas the P4Info metadata is target-agnostic and applied by the controller for API calls on top of defined tables and external instances.

### B. ONOS Controller

The Open Network Operating System (ONOS) [9] is an SDN controller developed by the Open Networking Foundation. This open-source controller is modular, extensible, and distributed. It provides operating system-like functionalities to the network from resource allocation to user interface applications in order to provide end-to-end network management and control solutions. ONOS is one of the few open-source controllers that support P4RT as well as OF. It was originally designed to work with OF-based switches but later extended to support programmable data plane devices such as P4-based devices. This extension allows users to use their own forwarding pipelines and applications to control devices with programmable data planes.

Fig. 1 depicts the main architectural units of ONOS, highlighting those added to enable P4RT support for controlling devices with programmable data planes. The different tiers of functionalities of ONOS are described in the following:

- **Top layer Applications:** Custom applications for controlling data planes reside in this layer. There are two types of applications: *(i)* Pipeline-agnostic applications such as Reactive Forwarding, where existing OF-based applications can be reused to control any P4-defined pipeline; *(ii)* Pipeline-aware applications that allow controlling custom/new protocols as defined in the P4 program.
- **Core:** This central tier contains the network state logic and access to the network functions. It communicates with the application layer via the NorthBound (NB) API and with the lower layers via the SouthBound (SB) API. The Pipeconf store application residing in this layer is added to support P4RT, where it packages all files necessary to let ONOS understand, control, and deploy an arbitrary pipeline. The translation services take care of translating pipeline-specific entities from protocol-dependent representations to Protocol-Independent (PI) ones.
- **Driver/Provider:** Applications in this tier represent a family of data plane devices to ONOS. A device-specific logic is confined in these components with a level of abstraction allowing applications to interact with this specific family of devices.
- **Protocol:** SB protocols like OF and P4RT run in this layer. The encoding and decoding of the messages and packets being received or sent by ONOS take place in the applications residing in this layer.

### C. Related Work

To the best of our knowledge, there is no prior work on evaluating the performance of P4RT controllers. The only related work in terms of benchmarking builds on top of OF controllers. Due to the maturity of OF, there is indeed abundant literature in this regard, e.g., [2], [14]–[21]. The main focus is to evaluate the performance in terms of latency and throughput, and it has been concluded that the performance highly depends on the threading capability of the controller, which shows the number of requests it can process in a single time slot [2]. In addition, reliability and energy consumption are evaluated in some works [16], [21].

We build our new P4RT benchmarking tool on top of OFCProbe [10], which is originally developed for benchmarking OF controllers. OFCProbe can run on any platform, scale according to the available computing resources (e.g. cores), provide detailed statistics, and follow a modular implementation. In this work, we leverage the modular design of OFCProbe for extending it to support benchmarking P4RT controllers.

## III. P4RCProbe Implementation

In this section, we describe the design and implementation of P4RCProbe, a novel tool for benchmarking P4RT controllers.

As the literature is rich with open-source benchmarking tools for OF controllers, we decided to build on top of these tools to leverage their mature development stage. Accordingly, we extend the OFCProbe tool, which was designed originally for benchmarking OF controllers, to support benchmarking P4RT controllers. The modular design of OFCProbe enables the smooth extension/integration of P4RT-enabler modules to it. Fig. 2 shows the original OFCProbe architecture along with the modified components for enabling benchmarking P4RT controllers.

In a nutshell, the tool emulates dummy- or virtual switches that each initiate a connection with SDN controllers. Then, these virtual switches send Packet-In messages to the controller and receive the Packet-Out responses. The tool records

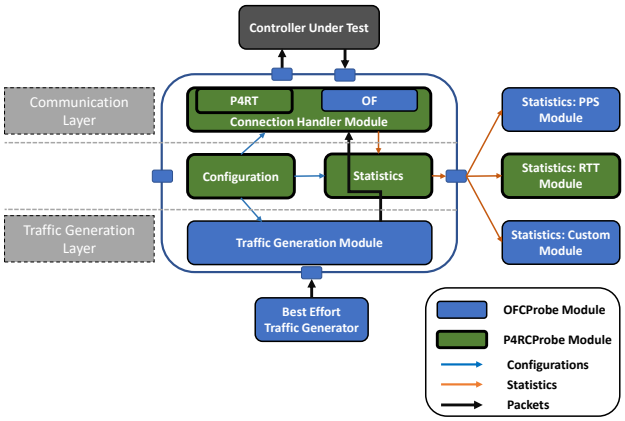



Figure 2: P4RCProbe Architecture.

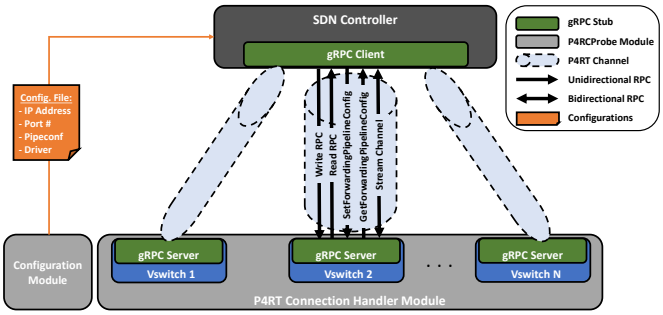



Figure 3: Communication Channels between P4RCProbe and P4RT-based SDN controllers.

statistics related to the controller's supported packet rate, RTT, etc. While the tool's Communication Layer is responsible for establishing and managing the connection channel between the virtual switches and the SDN controller, the Traffic Generation Layer manages the tasks related to the life cycle of packets to be sent by the virtual switches such as packet generation, handling, queuing, scheduling, etc. In the following, we describe the updated implementation of the benchmarking tool, highlighting the modified modules.

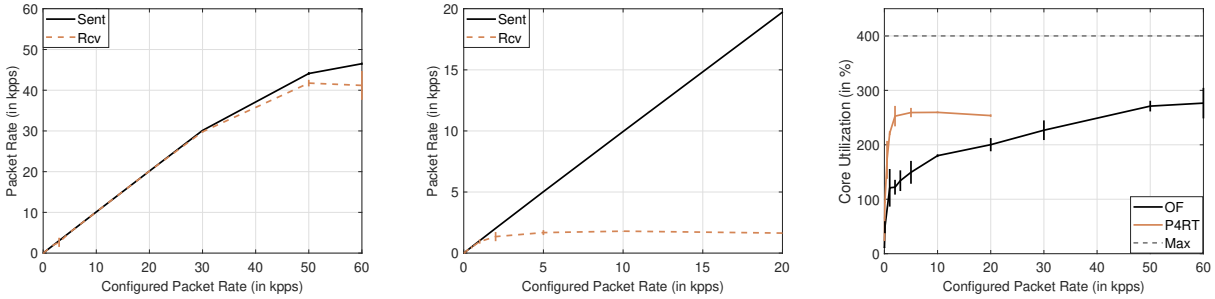
*1) Connection Handler Module:* The communication framework adopted in P4RT is based on Remote Procedure Calls (RPC) using gRPC and Protocol Buffers. This paradigm is different from that adopted in OF, which is based on the OF protocol. To support both types of connections, a parent connection handler module class is defined in P4RCProbe, where OF and P4RT connection handlers implement/inherit it. The P4RT connection handler instantiates a gRPC server stub according to the P4Runtime protocol buffer file "P4Runtime.proto" [22] for every switch to be emulated using the tool. All server stubs communicate with the P4RT client stub running in the SDN controller. For each P4RT client-server communication channel, the following 5 basic RPCs are established and managed:

- **Write RPC**: a unidirectional RPC used to update one or more P4 entities on the target.
- **Read RPC**: a unidirectional RPC used to read one or more P4 entities from the target
- **SetForwardingPipelineConfig RPC**: a unidirectional RPC used to set the P4 forwarding-pipeline config on the target.
- **GetForwardingPipelineConfig RPC**: a unidirectional RPC used to get the current P4 forwarding-pipeline config running on the target. The "Cookie field", i.e., equal to the hash function of the configured pipeline used to uniquely identify forwarding pipelines, is filled in the messages of this RPC by the tool after extracting the value of this field from SetForwardingPipelineConfig message. This way, the tool emulates for ONOS that all instantiated switches run a specific forwarding pipeline.
- **StreamChannel RPC**: a bidirectional stream between the controller and the data plane device (initiated by the controller). It is used to initiate a connection through client arbitration, checking switch session liveliness, streaming of notifications from the switch, and most importantly communicating Packet-In and Packet-out messages between the switch and the controller. P4RCProbe keeps sending Packet-In messages over this channel with TCP SYN as a payload, and metadata that is consistent with the definition of Packet-In headers in the adopted P4 pipeline such as the ingress port, padding, etc. Packet-Out messages received from the controller on this channel are decoded to update the statistics.

When P4RCProbe is configured to run multiple switches, it instantiates multiple gRPC server stubs that operate independently and communicate with the same client stub implemented in the controller as shown in Fig. 3.

*2) Configuration Module:* Different parameters can be passed to the tool to customize the benchmarking procedure. For example, a user can specify the runtime protocol to be used (OF or P4RT), the number of devices to emulate, the number of packets to be sent per switch, the desired statistics modules to be activated, etc. These parameters are filled in a configuration file and then used to customize different modules in the tool. In case the runtime protocol is selected to be P4RT, the controller needs to know which P4 data plane forwarding pipeline is used. P4RCProbe automatically generates a configuration file and passes it to the controller containing all necessary fields for initializing the controller. This information includes the selected pipeline configuration file, device driver (e.g. BMv2), ID of different switches, IP address, and port of different switches. The latter should be provided to the controller, as the client stub runs on the controller in the P4RT framework, and thus, it is the party that will initiate the connection.

*3) Statistics Module:* The RTT statistics module is extended in the tool to be compatible with the P4RT mode. RTT is calculated as the difference between a timestamp taken when the Packet-In message leaves the benchmarking tool from another timestamp taken when the corresponding Packet-Out message is received. In OF mode, packets are uniquely




(a) Sent/ Received rate for OF mode. (b) Sent/ Received rate for P4RT mode. (c) Core utilization for OF & P4RT modes.

Figure 4: Evaluation results for single switch case.

identified based on the "Transaction ID" field defined in the OF protocol. In P4RT mode, the unique ID is rather encoded in the payload of the randomly generated TCP SYN Packet-In messages.

Fig. 3 shows the interaction between P4RCProbe and the SDN controller detailing the different established communication channels. The source code for P4RCProbe [1] is made publicly available.

## IV. Evaluation

In this section, we use P4RCProbe to benchmark the performance of the ONOS 2.5.0 release when running in P4RT mode. The evaluation is also conducted when ONOS runs in OF mode, to have a baseline for assessing the efficiency of the P4RT-based implementation.

A testbed is built to conduct this evaluation. It is made up of two connected machines running Ubuntu 18.04, and equipped with a CPU with 4 cores (Intel i5-4670 at 3.40GHz), and two 10 Gbps Ethernet ports. P4RCProbe runs on one machine while ONOS controller, which is the entity under test, runs on the other machine. We use the open-source project "P4tutorial", available in the ONOS repository, as the P4 data plane in this evaluation. The project includes a P4 program that implements basic forwarding and tunneling functions, as well as the corresponding configuration files and the interpreter file that enables ONOS to understand the specific constructs of the given P4 program. This P4 pipeline, like OF switches, can interact with the "Reactive Forwarding" application on ONOS that controls the packet forwarding. The latter is important to guarantee a fair comparison of the performance of ONOS when running in P4RT vs OF mode while running the same control application, i.e. Reactive Forwarding. Each measurement is repeated 5 times where ONOS service is restarted between runs, and confidence intervals are plotted.

The case when a single switch connects to ONOS is evaluated first, then we extend the evaluation to the case of multiple switches.

### A. Single Switch Evaluation

In the following, we intend to have a first-hand understanding of the performance of ONOS in handling incoming packet streams. For this purpose, we configure P4RCProbe to mimic a single P4RT switch to be connected to ONOS running in P4RT mode, while varying the configured packet rate to be sent to the controller. The same evaluation is conducted for the OF case.

Different evaluated metrics for the single switch case are shown in Fig. 4. Fig. 4a shows the Packet-In sent rate to ONOS as well as the Packet-out received rate from ONOS as a function of the configured rate to be sent by P4RCProbe. It can be observed that ONOS in OF mode can handle all incoming packets up to 30k pps. After this point, packet drop starts taking place until the received packet rate saturates at around 42k pps. Unlike the OF case, the results corresponding to P4RT, shown in Fig. 4b, reveal that ONOS can only handle around 1.5k pps when operating in P4RT mode. Fig. 4c shows the core utilization of ONOS when running in OF and P4RT modes in response to an increasing incoming packet rate. Looking at low configured packet rates, it can be observed that ONOS processing in OF mode requires less computing resources when compared to P4RT. Moreover, while the OF mode processing shows an increasing scaling behavior, the P4RT mode saturates again at around 250 % core utilization when the packet rate reaches around 1.5k pps. Recalling that ONOS is running on a machine with 4 cores, the maximum achievable core utilization is 400 %, and thus there is no hardware processing limitation.

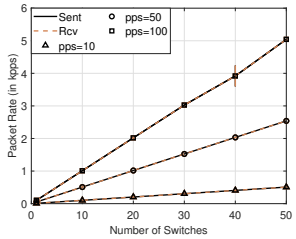
Note that as the purpose of this evaluation is to find the limits of ONOS in handling increased packet rates, the RTT statistics module is disabled in this evaluation to avoid overwhelming the tool with stored timestamps of packets without response.

### B. Multiple Switch Evaluation

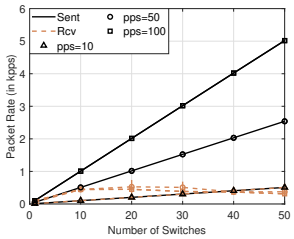
In this section, the tool is configured to mimic an increasing number of switches connected in a network and controlled by ONOS. The evaluation is again conducted when ONOS runs in OF and P4RT modes. Guided by the results of the previous subsection, the configured packet rate to be sent by every switch is selected such that the aggregated rate sums up to values around the 1.5k pps limit.

Fig. 5 shows the aggregated packet rate sent to and received from ONOS running in OF and P4RT modes as a function of

[1] https://github.com/tum-lkn/P4RCProbe

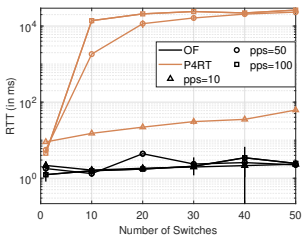



(a) OF.

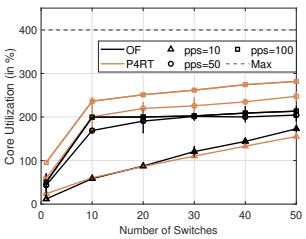



(b) P4Runtime.

Figure 5: Send and received rate for multiple switches case.




(a) RTT results.




(b) Core results.

Figure 6: Evaluation results for multiple switches case.

the configured number of switches when every switch sends 10, 50, and 100 pps. It can be observed that ONOS running in OF mode handles all incoming traffic smoothly as the number of switches and packet rates per switch increase up to 5000 pps (= 50 switch * 100 pps), where sent and received rates are overlapping. In P4RT mode, ONOS can handle packets from all switches as long as the sent packet rate per switch is 10 pps. However, when the packet rate per switch increases to 50 pps and 100 pps, the received packet rate saturates after 10 and 1 switch respectively. Interestingly, this saturation occurs at a rate less than 1000 pps (20*50 pps), which is lower than the previously identified limit in the single switch evaluation. This is due to the extra processing required by ONOS to handle multiple connections with more than one switch.

Fig. 6a shows the recorded average packets' RTT in ms and in logarithmic scale as a function of the configured number of switches for different per switch packet rates and when ONOS runs in OF and P4RT modes. In general, the RTT in OF mode varies between 1.5 and 4.3 ms with a slightly increasing trend as the number of switches increase. The configured per switch packet rate has a minimal impact on the RTT as the processing load in this evaluation is considered low for ONOS in OF mode, which can steadily process up to 30k pps. Looking to P4RT mode results, the RTT values sharply increase. In the stable case with 10 pps per switch, the RTT value increases from 8.9 to 61 ms as the number of switches increases to 50. For 50 pps per switch, the RTT increases to 1.8 seconds at 10 switches and then it reaches around 20 seconds at 50 switches. The RTT values corresponding to the 100 pps case directly reach 20 seconds for more than 1 switch.

Fig. 6b shows the core utilization of ONOS when running under OF and P4RT modes as the number of switches increase for different per switch packet rate configuration. For low packet rate (10 pps), both P4RT and OF modes require similar core resources, which increase from 10 to 170 % as the number of switches increases to 50. When the packet rate increases to 50 and 100 pps, the required core resources in both modes increase. However, P4RT consumes more core resources compared to OF reaching utilization of around 200 and 280 % at 50 switches for 50 and 100 pps respectively, while OF requires around 200 % for both cases. Again, no hardware limit is reached in this evaluation as the core utilization never reaches 400 %.

Another relevant statistic, which can be evaluated by P4RCProbe is the number of outstanding packets, which is the number of Packet-In messages sent to ONOS without an answer. Fig. 7 shows the number of outstanding packets when ONOS is running in P4RT mode, and the number of switches is set to 50 over a 60 seconds period. When the per switch packet rate is set to 10 pps, shown in Fig. 7a, the number of outstanding packets is always smaller than one as ONOS can handle all incoming traffic as recorded from Fig. 5b. More interestingly is the case where ONOS can not handle all incoming packets as shown in Fig. 7b and 7c for rates equal to 50 and 100 pps per switch respectively. The number of outstanding packets in these cases grows linearly over time until reaching around 2550 packets at 60 seconds for each switch when the packet rate per switch is set to 50 pps, and around double that value (5500 packets) when the rate is set to 100 pps. This shows that consistent buffering behavior is taking place in ONOS running in P4RT mode, where packets have to wait until they get served. This fact is further backed up by the observation that all packets sent to ONOS were received back after some time (tens of seconds) without any packet drop.

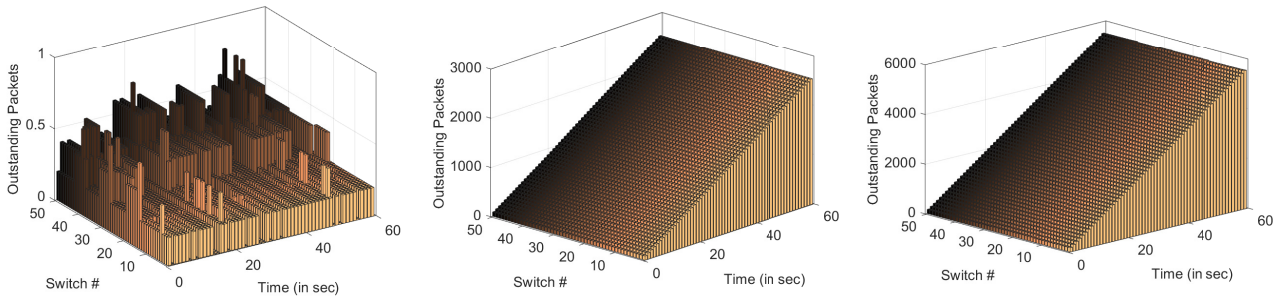
On the positive side, looking at the variation of the outstanding packets across different switches, it can be concluded that ONOS deals fairly with all switches since the accumulation of outstanding packets is consistent across switches. Note that the results corresponding to OF mode are not shown as they are always less than one since ONOS can handle all incoming packets while running in this mode.

## V. Using P4RCProbe to optimize ONOS

After observing the moderate performance of ONOS in P4RT mode, we showcase how P4RCProbe could be used to identify bottlenecks in controller implementations. Furthermore, we propose a possible design optimization for mitigating an identified performance bottleneck.

Our approach is as follows. We artificially create a shortcut after each processing block in ONOS as shown in Fig. 8. A shortcut is created to forward packets back to P4RCProbe directly after the gRPC Client, the P4Runtime protocol decoding module, and the reactive forwarding application to create paths P1, P2, and P3 respectively.

For each path, P4RCProbe is used to measure the received packet rate (successfully processed by ONOS without drop) when 10k pps are sent. Each measurement is conducted five




(a) Configured packet rate set to 10 pps. (b) Configured packet rate set to 50 pps. (c) Configured packet rate set to 100 pps.

Figure 7: Recorded outstanding packet in P4RT mode for different configured packet rates.

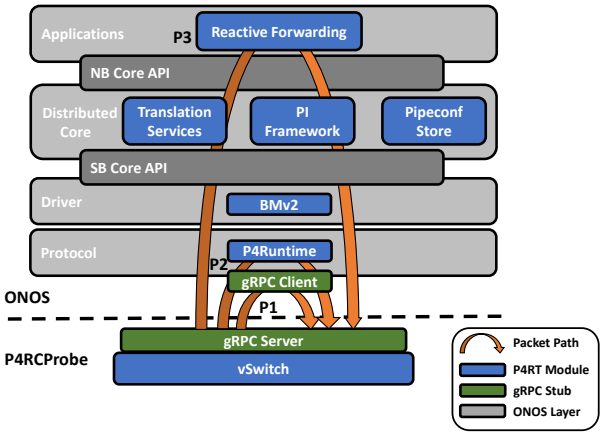



Figure 8: Method applied for identifying bottlenecks in ONOS controller running in P4RT mode.

times to record the average and standard deviation of the packet rate and core utilization. While the received rate on Path 1 after the gRPC client processing is observed to stay at around 10k pps, this rate directly drops to around 1.8k pps on Path 2 after the P4Runtime protocol module processing. This clearly indicates that there is a bottleneck at this stage. The received rate in Path 3, where the complete processing in ONOS is done, is found to be equal to around 1.5k pps.

Next, the method which implements the P4Runtime protocol processing in ONOS source code is analyzed to identify the reason for this bottleneck. We observed that ONOS retrieves the pipeline configuration object "PipeConf" for every received packet. This PipeConf object contains information describing the loaded P4 data plane on the switch including JSON files. Getting this PipeConf object that includes a method for reading input streams from stored files, is proven to be processing-intensive, especially after observing that the received rate increases again to 10k pps when skipping the corresponding lines of code. Recalling that the P4 data planes can be reprogrammed, getting the PipeConf object for every packet is meant to make sure that each packet gets processed according to the latest loaded P4 data plane.

One possible design optimization is suggested and implemented to mitigate this bottleneck. Instead of reading the PipeConf object for every packet, it is only read once when the client gRPC stub is created to get the current loaded P4 data plane. Then, a signal is sent to read the PipeConf object only when the data plane on the P4 device is changed. The latter can be detected by the GetForwardingPipelineConfig RPC, where the latest running P4 pipeline is always checked. This way, the heavy processing call for getting the PipeConf object is executed only once when the P4 data plane is modified instead of taking place for every packet. After applying this code patch, the packet rate that could be handled by the P4Runtime protocol block on Path 2 increases again to around 10k pps, indicating that this bottleneck is mitigated. Table I summarizes the recorded received packet rates and core utilization (average and standard deviation) for different packet processing paths before and after implementing the design optimization. The core utilization results are in line with the packet rate observations. This utilization decreases from 167 % to 66 % over Path 2 after applying the code patch, indicating a relaxed processing load at this stage.

Table I: Packet rate and core utilization recorded on different paths before and after applying the code patch.

| Metric | Code \ Path | P1 | P2 | P3 |
|---|---|---|---|---|
| *Rate (PPS)* | Original | 10030 ± 10 | 1836 ± 41 | 1471 ± 10 |
| | Enhanced | 10026 ± 8 | 10050 ± 7 | 1723 ± 30 |
| *Core (%)* | Original | 61 ± 0.2 | 167 ± 0.7 | 215 ± 1.6 |
| | Enhanced | 60 ± 0.1 | 66 ± 0.2 | 206 ± 1.1 |

Note, that the packet rate on Path 3 after the complete ONOS processing drops to 1723 pps with the design optimization implemented, which is still more than the rate on this path before implementing the optimization, i.e. 1471 pps. Meaning that although this design optimization solved the processing bottleneck in the P4Runtime protocol block, there exist other bottlenecks at a later stage within the ONOS processing pipeline. Nevertheless, solving this bottleneck could improve the overall processing rate of ONOS in P4RT mode by around 17 %. Solving the remaining bottlenecks of ONOS is out of the scope for this paper, as the purpose here is to show how our proposed benchmarking tool could be used in practice for identifying bottlenecks in the design of SDN controllers.

## VI. CONCLUSION

The controller is a crucial component in the SDN architecture, and its performance has a significant impact on the overall performance of the network. Thus, benchmarking and evaluating the performance of the controller has been done in the past but mainly limited to OF-based controllers. In this work, we fill the gap by providing the first benchmarking tool for P4RT-based controllers called P4RCProbe. A comprehensive evaluation of the ONOS controller when running in both OF and P4RT mode is conducted. Results show that while the P4RT implementation in ONOS treats all connected switches fairly, it has a lower packet processing rate and longer RTT values compared to OF. The latter demonstrates the overhead on the control plane to interact with a flexibly programmable data plane. A method to use P4RCProbe and identify bottlenecks in P4RT-based controllers is then proposed and applied to mitigate a bottleneck in the ONOS P4RT implementation showing a 17 % improvement in the average packet processing rate.

The tool can be used to further debug and enhance the performance of the ONOS P4RT implementation. Moreover, the tool could be used to benchmark the performance of upcoming SDN controllers that support controlling programmable data planes using P4RT.

## REFERENCES


[1] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[2] L. Zhu, M. M. Karim, K. Sharif, C. Xu, F. Li, X. Du, and M. Guizani, "SDN controllers: A comprehensive analysis and performance evaluation study," *ACM Computing Surveys (CSUR)*, vol. 53, no. 6, pp. 1–40, 2020.

[3] H. Harkous, M. Jarschel, M. He, R. Priest, and W. Kellerer, "Towards understanding the performance of P4 programmable hardware," in *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 2019, pp. 1–6.

[4] H. Harkous, M. Jarschel, M. He, R. Pries, and W. Kellerer, "P8: P4 with predictable packet processing performance," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 2846–2859, 2021.

[5] H. Harkous, M. He, M. Jarschel, R. Pries, E. Mansour, and W. Kellerer, "Performance study of P4 programmable devices: Flow scalability and rule update responsiveness," in *2021 IFIP Networking Conference (IFIP Networking)*, 2021, pp. 1–6.

[6] M. Helm, H. Stubbe, D. Scholz, B. Jaeger, S. Gallenmüller, N. Deric, E. Goshi, H. Harkous, Z. Zhou, W. Kellerer, and G. Carle, "Application of network calculus models on programmable device behavior," in *2021 33rd International Teletraffic Congress (ITC 33)*, Avignon, France, 2021.

[7] H. Harkous, N. Kröger, M. Jarschel, R. Pries, and W. Kellerer, "Modeling and performance analysis of P4 programmable devices," in *2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2021, pp. 1–7.

[8] A. M. D. Tello and M. Abolhasan, "SDN controllers scalability and performance study," in *2019 13th International Conference on Signal Processing and Communication Systems (ICSPCS)*, 2019, pp. 1–10.

[9] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "ONOS: Towards an open, distributed SDN OS," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 1–6.

[10] M. Jarschel, C. Metter, T. Zinner, S. Gebert, and Phuoc Tran-Gia, "OFCProbe: A platform-independent tool for OpenFlow controller analysis," in *2014 IEEE Fifth International Conference on Communications and Electronics (ICCE)*. IEEE, 2014, pp. 182–187.

[11] "P4Runtime Specification v1.2.0," https://opennetworking.org/wp-content/uploads/2020/10/P4Runtime-Specification-120-wd.html, accessed: 2021-07-15.

[12] "Protocol Buffers," https://github.com/protocolbuffers/protobuf, accessed: 2021-07-15.

[13] "gRPC," https://grpc.io/, accessed: 2021-07-15.

[14] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in Software-Defined Networks," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*. USENIX Association, 2012, p. 10.

[15] M. Jarschel, F. Lehrieder, Z. Magyari, and R. Pries, "A flexible OpenFlow-controller benchmark," in *2012 European Workshop on Software Defined Networking*, 2012, pp. 48–53.

[16] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of SDN/OpenFlow controllers," in *Proceedings of the 9th Central Eastern European Software Engineering Conference in Russia*. ACM, 2013.

[17] Z. K. Khattak, M. Awais, and A. Iqbal, "Performance evaluation of OpenDaylight SDN controller," in *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2014, pp. 671–676.

[18] L. Andrade, M. Borba, A. Ishimori, F. Farias, E. Cerqueira, and A. Abelém, "On the benchmarking mainstream open Software-Defined Networking controllers," in *Proceedings of the 9th Latin America Networking Conference (LANC)*. ACM, 2016, p. 9–12.

[19] S. Rowshanrad, V. Abdi, and M. Keshtgari, "Performance evaluation of SDN controllers: Floodlight and OpenDayLight," *IIUM Engineering Journal*, vol. 17, no. 2, p. 47–57, Nov. 2016.

[20] A. Nguyen-Ngoc, S. Raffeck, S. Lange, S. Geissler, T. Zinner, and P. Tran-Gia, "Benchmarking the ONOS controller with OFCProbe," in *2018 IEEE Seventh International Conference on Communications and Electronics (ICCE)*. IEEE, 2018, pp. 367–372.

[21] S. Mallon, V. Gramoli, and G. Jourjon, "Are today's SDN controllers ready for primetime?" in *2016 IEEE 41st Conference on Local Computer Networks (LCN)*. IEEE, 2016, pp. 325–332.

[22] "P4runtime.proto," https://github.com/p4lang/p4runtime/blob/main/proto/p4/v1/p4runtime.proto, accessed: 2021-07-15.