BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2018
http://excel.fit.vutbr.cz

Configuration of Remote P4 Device

Jakub Neruda*

Abstract
Administration of a large network from a central node with a vendor independent API is quite important issue these days. The SDN concept was truly helpful with realization of the solution, namely in the form of the OpenFlow protocol. Nowadays, a P4 language is gaining momentum, primarily thanks to its ability to describe whole packet processing pipeline and also for the P4 Runtime, which provides a solution to the distributed network configuration. CESNET association is one of the research groups starting to support P4 in their network cards of the Combo series. The aim of this work was to make configuration of the Combo cards from the P4 Runtime compatible solutions possible. An API was designed for these cards, aimed at the dynamic flow table configuration and control. With this API, a support in the P4 Runtime was implemented and the card now can be configured from the P4 ready SDN applications.

Keywords: P4 — SDN — P4 Runtime

Supplementary Material: GitHub Repository

*xnerud01@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

The complexity and size of networks is growing every day, requiring more sophisticated solutions for their configuration to be developed. The demand of the market moved from the basic solution for setting up firewalls and QoSes to solutions that allow whole network reconfiguration on the fly, as well as experimenting with brand new network protocols. The first promising technology was OpenFlow that not only implemented standard SDN features but also allowed for a limited network device programming through a standardized set of known packet header fields[5].

Nowadays, the P4 language is getting a lot of attention as it is successor to OpenFlow, because programmers can describe the whole packet processing pipeline in an abstract, platform independent language, significantly reducing the amount of extra work coming from working with devices from multiple manufacturers [4]. As the P4 as a language is pretty much established concept by now, its incorporation into SDN ecosystem had only just begun. And as more organisations are looking into the development of P4 compatible devices, the need for universal configuration mechanism is greater than ever[2].

The same motivation applies for the CESNET organisation that developed a network card Combo-100G2Q. This card is programmable in the P4 language, thanks to the reconfigurable FPGA chip it is equipped with[3]. This paper describes the process and the challenges of incorporating mentioned card into what is known as the P4 Runtime ecosystem – the SDN solution for P4 compatible devices.

2. Problem

At the time of writing, the mentioned card supports only the $P4_{14}$, but the support for newer language re-

vision $P4_{16}$[1] is being developed. Supported features are currently limited to the most important feature of the $P4_{14}$ language – rule tables (the support of registers and counters is currently in testing phase). These entities are defined in the P4 program, but their contents are defined during the runtime by a user. Thanks to this, the device can be dynamically adapted to fulfill needs of the network administrator.

Initially, the Combo card had only very rudimentary API for filling these tables with rules, written in the C language. The biggest challenge in using this API was in pushing the rules into the tables in the device itself. This was because all the rules had to be stored in the continuous memory block and inserted into the device at once, modifying all tables in a single call, clearing anything that has been stored there before.

This is sufficient for use cases where contents of the tables are to be modified only a handful of times, but quite impractical in any long running, persistent application. In such cases, gradual updates of the tables are more common as the device is being trained how to behave on the fly. So if the existing API has only means of creating the rules and inserting them into the tables all at once, a new solution has to be developed to allow per-rule alteration like modifying and deleting those rules.

This is also the most basic prerequisite for integrating the card into any SDN ecosystem since the SDN apps work exactly in the way mentioned above.

2.1 Metodology

Currently, the ONOS[2] is the only known implementation of SDN controller which is capable of working with P4 Runtime. This project is based on specifications created by the P4 API Work Group led by Antonin Bas from Barefoot Networks[1] and was already demonstrated at the SDN NFV World Congress [6]. The specification of the P4 Runtime is developed under the PI repository[3] and contains example implementations of device support on the lowest level of the SDN architecture.

As the P4 Runtime establishes a known, unified interface, any P4 device must be able to understand this interface to ensure the compatibility with any existing or future SDN controller. Thanks to that, the interface can be used as well to design new control API for the Combo card. Please note that newly designed API

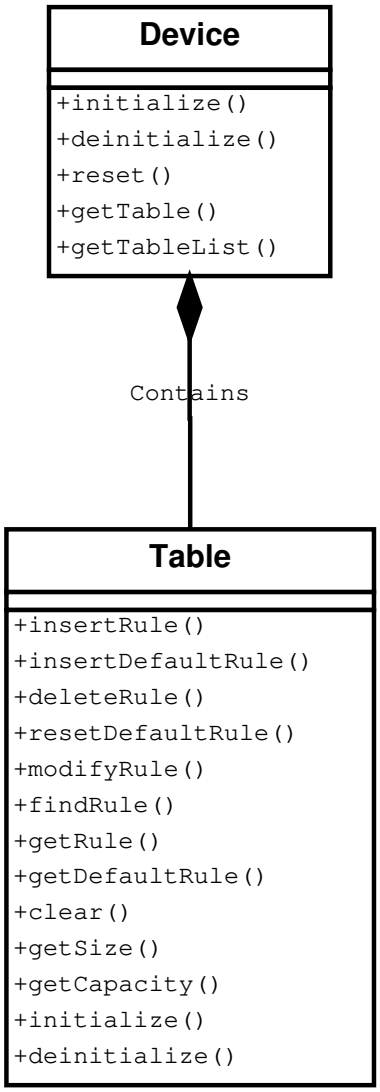

Figure 1. UML for Combo card control API

still relies on the previously established methods for rules creation, but it adds an abstraction over a device we're connecting to and also an abstraction over tables inside the device. The UML diagram of the API is shown in the Figure 1.

As you can see, after the connection is established with a certain device (via `initialize` method), a programmer can request the device for a table handle which is then used for a manipulation with rules in the associated table. All memory related operations are hidden from the user and tables can be conveniently modified on a per-rule basis. Also, thanks to slight modifications in the old API, tables no longer have to be updated all at once, but only all rules within a single table have to be modified at once. This lessened the runtime overhead and greatly reduced the complexity of actual implementation.

Although persistent applications can now be easily implemented with this new API, there are few downsides to it as well. The major issue comes from

[1] https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html

[2] https://wiki.onosproject.org/display/ONOS/P4+Runtime+support+in+ONOS

[3] https://github.com/p4lang/PI

the hardware implementation of P4 in the Combo card which does not provide means to retrieve rules already stored in the card at the time of establishing connection to it. So even though any configuration application can be persistent, any restart will also reset the state of the tables within a device. Second downside comes from the used language. Original API was written in C, whilst the new one was written in C++, due to its OOP design. Therefore, the user is currently forced to use two APIs, one for creating the rules and second for inserting them into the tables. However, this issue is only temporary since plans for unifying both APIs under C++ version are already in discussion.

3. P4 Runtime

With the Combo card API developed to match the requirements of the P4 Runtime, the incorporation into the P4 SDN ecosystem was yet to be implemented. As was said earlier, the specification of the architecture is called P4 Runtime. The main aim of P4 Runtime specification is to cover how the SDN controller should communicate with the maintained devices. A simplified schema of the components involved in the whole process is shown in the Figure 2. The figure also highlights the parts which are in the scope of this work.

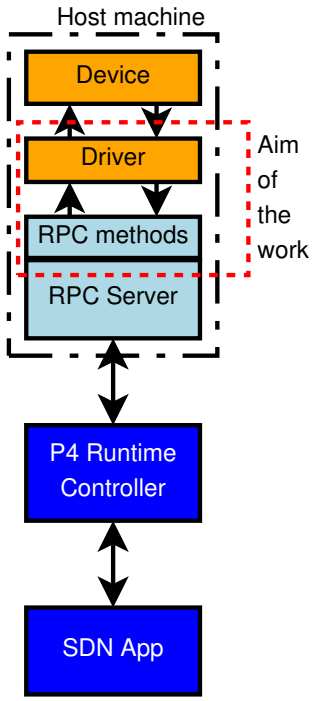

Figure 2. P4 Runtime communication pipeline

As you can see, the SDN app communicates with the dedicated controller. Both components – the controller as well as SDN app – are out of scope of the P4 Runtime specification and any developer can implement it in any way he sees fit. At the moment, there is the only one publicly available project, the ONOS project mentioned earlier.

Once the administrator enters a command in his SDN App, this command is propagated into the controller and then it is sent to managed devices using a Remote procedure call (RPC). Data for RPC calls are serialized using the Google's protobuf[4] technology and a special metadata file called P4 Info. The P4 Info maps names defined in P4 program into unique identifier numbers and also dictates bytewidth of each element.

Command and its data are delivered to the machine hosting the network device where a Google gRPC[5] server is running. The appropriate user implemented RPC code is then executed, deserializing the data and calling some sort of driver layer between host machine software and network device hardware, propagating new data into the device. Appropriate response is then returned through the gRPC into the controller and the SDN app.

In this architecture, the RPC methods must be implemented in such way that utilizes the driver layer (new API in this case). When implementing the bodies of RPC methods, the bmv2 target[6] was used as a reference, due to lack of a proper documentation from the authors of P4 Runtime. Simply put, an implementation of any RPC call can be divided into three major steps which differ based on the requested action. First, the programmer has to obtain a device handle and a P4 Info data. RPC call only provides IDs for those entities, the programmer has to come up with his own system for storing and retrieving them. Then, in case of a write request, the incoming data have to be deserialized using the P4 Info. When the data are parsed, the programmer can call his driver layer to perform the requested action. If the action was a read request, the obtained data must be serialized, once again using the P4 Info to perform the task.

With the methods implemented, one can compile the project which will yield a dynamic library. This library contains the implemented RPC methods and it can be used to run a gRPC server on the card's host machine. After that point, any P4 Runtime conforming controller can connect to it and manage card's tables (and more in the future). To demonstrate that ability, a demo controller implementation from PI repository[7] was modified and used. A Figure 3 shows a screenshot of a very simple SDN App implemented as a web page

[4] https://github.com/google/protobuf
[5] https://github.com/grpc/grpc
[6] https://github.com/p4lang/PI/tree/master/targets/bmv2
[7] https://github.com/nerudaj/PI/tree/master/proto/demo_grpc

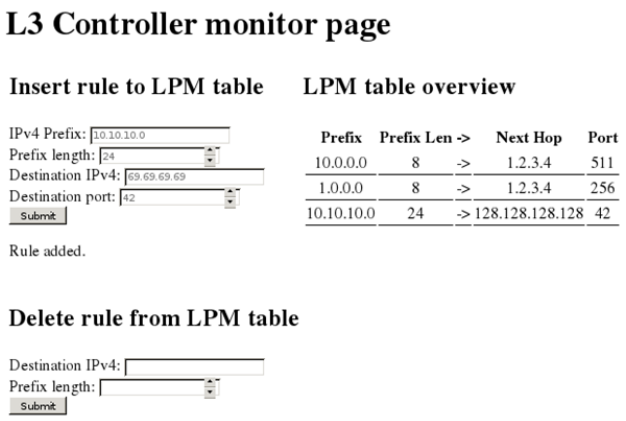

Figure 3. Screenshot of demo SDN application

and generated by the demo controller.

The original demo was capable of changing the P4 program in the device and reading a value of a counter defined in the P4 program. Both of these actions are not yet possible to do with the Combo card. The modified demo allows the user to access a table with keys using the Longest Prefix Match (LPM) matching engine and to insert or delete routing rules in that table. The controller remembers the changes made to the table in its own local memory block and displays the content of the LPM table to the user as well.

The demo is intended to be only a proof-of-work as its capabilities are very limited. Also, the original programmer had not built the demo built with future improvements in mind and thus modifying it is not trivial or viable. Fortunately, a user can use a CLI application that is also present in the PI repository[8]. The CLI works locally on the card's host machine, utilizing the same dynamic library with the RPC method implementations. It provides access to full P4 Runtime interface and for now it is the easiest way to test the new features.

4. Conclusion

Even though the P4 Runtime specification is still under heavy development and there is no comprehensive documentation at the moment, the incorporation of the Combo card family into P4 ecosystem was done and now the card can be easily controlled from any P4 compatible controller. There is still room for improvement in terms of API itself and number of features the card should support and it will be the main focus of the future development.

Although original API for controlling the Combo cards is not available to general public, the new API as well as an implementation of the RPC methods is released as open source software. Therefore, anyone can use it as a reference while implementing similar support for their own hardware[9].

[8] https://github.com/p4lang/PI/tree/master/CLI

[9] https://github.com/nerudaj/PI/tree/devel/targets/combo

Acknowledgements

I would like to thank my supervisor Pavel Benáček, Ph.D. for his support, guidance and valuable advice.

References

[1] Antonin Bas and Lorenzo Visicano. P4 api wg charter. https://p4.org/p4-spec/docs/P4_API_WG_charter.html.

[2] Antonin Bas and Lorenzo Visicano. Announcing p4 runtime, 2017. https://p4.org/api/announcing-p4runtime-a-contribution-by-the-p4-api-working-group.html.

[3] Pavel Benáček. *Generation of High-Speed Network Device from High-Level Description*. PhD thesis, Faculy of Information Technology, CVUT, Praha, 2016.

[4] Path Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nich McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 3(44):87–95, 2014. ISSN: 0146-4833.

[5] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 2(38):69–74, 2008.

[6] Nick McKeown, Timon Sloane, and Jim Wanderer. P4 runtime - putting the control plane in charge of the forwarding plane, 2017. https://p4.org/api/p4-runtime-putting-the-control-plane-in-charge-of-the-forwarding-plane.html.