Exploring Runtime Reconfigurability of P4 Data Plane

Mu He, Andreas Blenk, Arsany Basta, Wolfgang Kellerer
Technical University of Munich, Germany
{mu.he,andreas.blenk,arsany.basta,wolfgang.kellerer}@tum.de

ABSTRACT

The current trend of network programmability stimulates the endeavor to program the data plane with domain specific languages such as P4. In face of constant changes in user demands and traffic fluctuations, the programmability, if revealed within a short time frame, can lead to runtime reconfigurations and potentially herald network adaptations. Such ability of runtime reconfiguration should be explored and evaluated to fully leverage the capability of programmable networks. In this work, we present two approaches to assist P4's data plane runtime reconfiguration. We also design a management entity to facilitate the coordination of reconfiguration and the necessary state synchronization. Evaluation results show the migration of a stateful firewall between two forwarding nodes at runtime without an interruption of network operation.

CCS CONCEPTS

• **Networks** → *Network manageability*;

KEYWORDS

Programmable Data Plane, Network Function Adaptation

1 INTRODUCTION

As one of the advantages that P4 promises, data plane programmability allows the deployment of network functions and protocols with ease and efficiency [4, 10, 11]. Such programmability also contributes to reconfigurability, which implies the renewal of network functionality at runtime, giving rise to higher adaptability of the data plane in the face of constant flux in traffic volume, as well as in requirements and policies on the traffic [2, 9, 12]. For example, when a single network function instance is no longer sufficient to satisfy the current load, dynamic scaling can create new instances and split the load over the replicas [9]. Despite its merit, the runtime reconfiguration of the P4 data plane has not yet perceived much attention.

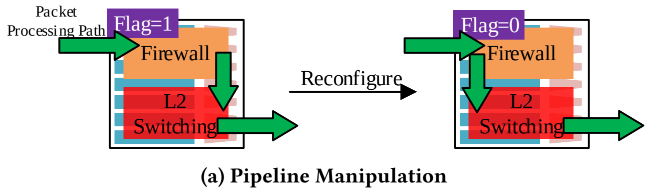
In this work, we verify the reconfigurability of P4 and evaluate the impact of reconfiguration on packet processing. The challenge is to design a mechanism that guarantees no service interruption during reconfiguration. We propose two approaches, namely *pipeline manipulation (PM)* and *program reload (PR)*, for updating the packet processing pipeline at runtime. The first approach leverages the binary *register*, which is a stateful variable defined in P4 specification, to represent if a network function is implemented in a certain node. The registers can be accessed and changed through the control/management API of the P4 target. The second approach directly reconfigures the data plane with new packet processing pipeline, which is suitable to more drastic reconfiguration of the data plane, e.g., protocol upgrade. Besides, a management entity (ME) is introduced to issue the reconfiguration-related control/management messages.

With the idea of stateful data plane, many stateful network functions, such as load balancer and web proxy, can be extracted from the SDN controller and implemented directly in the data plane, with the target of line rate processing [10]. Such trend renders a more challenging reconfiguration task. One part of the states, e.g. liveness of a flow, can be synchronized by piggybacking state update on live traffic within the data plane [5]. The other part, e.g., multipath routing decisions [7], need further split or merge before they can be synchronized, which calls for a delicate state management mechanism. To this regard, the ME we propose can assist the coordination of the state synchronization. As a proof-of-concept, we implement the two approaches on BMv2 target [8] and evaluate the performance in terms of the packet forwarding latency during reconfiguration. To the best of our knowledge, there has been no previous work in this regard.

2 RECONFIGURATION DESIGN

Data plane functions, e.g., L2 forwarding and firewall, can be written as P4 program snippets, and thereafter compiled and mapped into the Match+Action resources of the target,

CoNEXT'18, December 4, 2018, Heraklion, Greece
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6081-4/18/12…$15.00
https://doi.org/10.1145/3287153.3287155



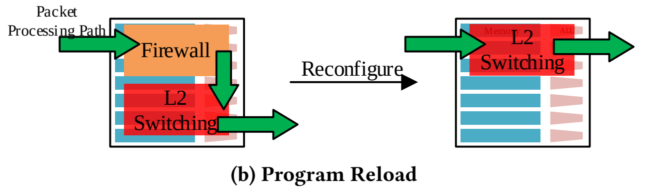


Figure 1: Two data plane reconfiguration approaches: (a) pipeline manipulation and (b) program reload.

e.g., a software [8] or a hardware switch with programmable switching ASIC [1].

Figure 1 demonstrates the Match+Action resource as the blue and pink blocks. L2 switching and firewall are logically represented as embedded resource, and the green arrows represent the packet processing path. We show the two approaches that remove the firewall from the forwarding node. *PM* uses the *register*: each function is guarded with a binary flag in registry. By updating the register values, the ME can control the path that packet processing traverses. After reconfiguration, the firewall block will be bypassed. *PR* addresses the reconfiguration from another perspective: compile new P4 code consisting of L2 switching only, send the configuration file via the control/management API, and force Match+Action resource remapping of the node. Notably, the remapping may cause data plane interruption for some targets, because unlike the former approach, the whole packet processing pipeline needs to be overwritten.

The ME also coordinates the state migration, by possessing a global view of the network state that resides in each node. Specifically, it decides what state to migrate and in which order the migration should proceed. Figure 2 shows the message exchange between the forwarding nodes and the ME when migrating a network function with *PR*. The *load_new_config_file* pushes the new compiled P4 program to the target forwarding node, followed by a series of *table_adds* which populate Match+Action tables. Message *register_reads* and *register_writes* transfer necessary network states from the source to the destination. The *swap_configs* message finally enables the new data plane. Such message exchange should be carefully designed to avoid inconsistent forwarding behavior.

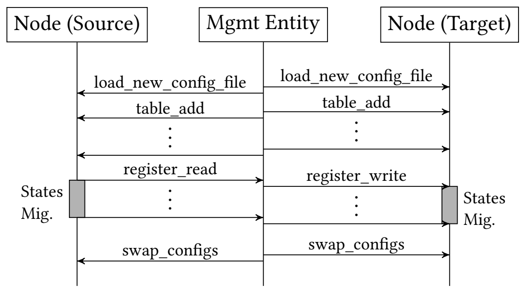


Figure 2: Message exchange during stateful data plane reconfiguration.

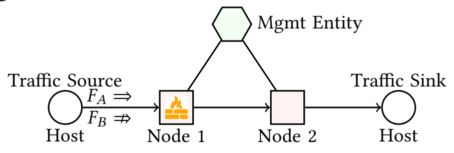


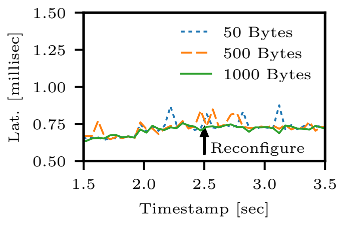
Figure 3: Experimental setup.

3 RESULTS AND FUTURE WORK

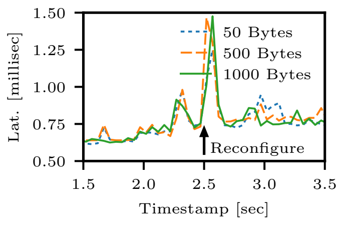
We assess the underlying data plane behavior for both approaches. As shown in Figure 3, we set up two forwarding nodes and migrate a stateful firewall from one node to the other node. The migration is coordinated by the ME written in Python. We generate two traffic flows F_A at 3k pps and F_B at 1k pps in parallel, and the firewall blocks F_B. Mininet [3] is applied to emulate the experiment network.

During the firewall migration, all packets of F_B are blocked. The state indicating if a flow should be blocked is properly migrated from source to target node. We variate the payload size of the competitive flow F_A and measure the forwarding latency repeatedly. Figure 4 shows that *PM* does not incur significant delay, whereas *PR* results in higher latency (up to 2x) for around 0.2 second. The better performance of *PM* comes at the cost of designing all desired functionalities in the beginning, which is not possible when NFs are gradually implemented and deployed. Because in P4 most computation effort is assigned to header processing, we observe similar curves of latency for different payload sizes.

We also implement the firewall as a software running inside a VM and deploy the VM in an OpenStack cloud environment. In this case, migrating the firewall, together with its state, is to live-migrate the VM between two physical servers. During migration, all packets of F_B are blocked; however, F_A experiences packet loss, which corresponds to a service interruption of around 200 ms. Different approaches have been proposed to eliminate such interruption, e.g., Split/Merge [9], OpenNF [6] and Dysco [12]. However, they assume general software NFs, which can be hard to customize. With P4 and the proposed approaches, we can create, deploy and reconfigure customized NFs in a more efficient manner without any data plane interruption.



(a) Pipeline Manipulation



(b) Program Reload

Figure 4: Data plane performance during reconfiguration for *PM* and *PR*.

For future work, we believe it would be interesting to evaluate the reconfiguration performance in large scale networks composed of P4 targets. Automating the process of sophisticated network state management during runtime reconfiguration can also be interesting to work on.

REFERENCES

[1] Barefoot Networks. 2018. Barefoot Tofino. (2018). https://www.barefootnetworks.com/products/brief-tofino/

[2] Adrian Caulfield, Paolo Costa, and Monia Ghobadi. 2018. Beyond SmartNICs: Towards a Fully Programmable Cloud. In *IEEE International Conference on High Performance Switching and Routing, ser. HPSR*, Vol. 18.

[3] De Oliveira et al. 2014. Using mininet for emulation and prototyping software-defined networks. In *IEEE COLCOM*. IEEE, 1–6.

[4] Katta et al. 2016. Hula: Scalable load balancing using programmable data planes. In *ACM SOSR*. ACM, 10.

[5] Luo et al. 2017. Swing State: Consistent Updates for Stateful and Programmable Data Planes. In *ACM SOSR*. ACM, 115–121.

[6] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. 2014. OpenNF: Enabling innovation in network function control. In *ACM SIGCOMM CCR*, Vol. 44. ACM, 163–174.

[7] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. 2013. B4: Experience with a globally-deployed software defined WAN. In *ACM SIGCOMM CCR*, Vol. 43. ACM, 3–14.

[8] P4 Language Consortium. 2018. P4-BMv2. (2018). https://github.com/p4lang/behavioral-model

[9] Shriram Rajagopalan, Dan Williams, Hani Jamjoom, and Andrew Warfield. 2013. Split/Merge: System Support for Elastic Execution in Virtual Middleboxes.. In *NSDI*, Vol. 13. 227–240.

[10] Anirudh Sivaraman, Alvin Cheung, Mihai Budiu, Changhoon Kim, Mohammad Alizadeh, Hari Balakrishnan, George Varghese, Nick McKeown, and Steve Licking. 2016. Packet transactions: High-level programming for line-rate switches. In *ACM SIGCOMM*. ACM, 15–28.

[11] Anirudh Sivaraman, Changhoon Kim, Ramkumar Krishnamoorthy, Advait Dixit, and Mihai Budiu. 2015. Dc. p4: Programming the forwarding plane of a data-center switch. In *SIGCOMM SOSR*. ACM, 8.

[12] Pamela Zave, Ronaldo A Ferreira, Xuan Kelvin Zou, Masaharu Morimoto, and Jennifer Rexford. 2017. Dynamic service chaining with dysco. In *ACM SIGCOMM*. ACM, 57–70.