Optimizing Data Plane Programs for the Network

Johannes Krude*, Matthias Eichholz†, Maximilian Winck*, Klaus Wehrle*, Mira Mezini†
*RWTH Aachen University, †Technische Universität Darmstadt
{krude,winck,wehrle}@comsys.rwth-aachen.de,{eichholz,mezini}@cs.tu-darmstadt.de

ABSTRACT

With the move of Software-defined networking from fixed to programmable data planes, network functions are written with P4 or eBPF for targets such as programmable switches, CPU based flow processors [5] and commodity CPUs [7]. These data plane programs are, however, limited in per-packet time budget [3] (e.g., 67.2 ns at 10GbE) and program size, making program optimization imperative [6]. Existing approaches focus on optimizing the distribution of flow rules in fixed data planes [4] or they are limited to a single switch [2]. We see great potential in integrating the network topology into program optimization.

CCS CONCEPTS

• **Networks → Programmable networks**.

KEYWORDS

programmable switches, eBPF, program optimization

ACM Reference Format:
Johannes Krude, Matthias Eichholz, Maximilian Winck, Klaus Wehrle, and Mira Mezini. 2019. Optimizing Data Plane Programs for the Network. In *NetPL '19: ACM SIGCOMM Workshop on Networking and Programming Languages, August 23, 2019, Beijing, China.* ACM, New York, NY, USA, 1 page. https://doi.org/10.1145/3341561.3349590

1 INTRODUCTION

Redundant computations by data plane programs on multiple nodes remain invisible to conventional compiler optimizations which only consider individual programs. In our approach, we incorporate the network topology to optimize the code of program chains. For example, the eBPF based Cilium load balancer [1] includes an extensive IPv6 option parser needed to locate the TCP header. If another node in the network unconditionally inserts a specific IPv6 option, the load balancer no longer needs to check whether this option is present. Our approach removes the no longer required parts of the parser, *increasing the packet rate of the Cilium load balancer by a factor of up to 2.3 and decreasing the compiled program size by 88%.* We envision our approach to be included in network controllers to optimize generalized data plane programs whenever needed.

2 CHAIN SELECTION

Parts of a data plane program are redundant if equivalent functionality was already applied to packets before reaching this program. The controller therefore uses its knowledge of the network topology to discover program chains with optimization opportunities.

Since the optimizations improve the performance without altering the resulting behaviour, they can be selectively applied to performance critical hotspots. In case such a hotspot is located at the intersection of multiple program chains, optimization opportunities can be increased through different variants of the same program, each optimized for a different program chain. Omitting programs without potential for optimizations from program chains reduces analysis time and increases reuse of optimizations without invalidating the optimizations.

3 CHAIN OPTIMIZATION

Our approach discovers optimizations for individual programs while considering the network topology. We therefore express a program chain as a combined program mimicking the network behavior, e.g., our example chain becomes:

```
if (insert_ipv6_options(&packet) != DROP)
        cilium_load_balancer(&packet);
```

On this combined program, exhaustive symbolic execution can prove whether branches are dead or parts of the packet are constant. Then, selectively replacing instructions in the individual programs, e.g., replacing dead branches with jumps in the cilium IPv6 option parser, enables further optimization with conventional compiler passes such as dead code elimination. Our prototype optimizes the cilium load balancer for our example chain in 30 s including 25 s of symbolic execution.

4 CONCLUSION & FUTURE WORK

We propose to apply optimizations to data plane programs which are valid for individual program chains. Our prototype shows significant optimization potential for real-world programs.

To reduce the costs of the symbolic execution, we plan to provide dedicated language abstractions to describe services and their compositions, which allow a less costly identification and removal of redundant code. By encapsulating header definitions in modules it becomes easier to identify identical headers across programs. When deriving conditions for headers their parsers can be minimized or even removed if e.g., a header is effectively always inserted.

Acknowledgements. Funded by DFG as part of CRC 1053 MAKI.

REFERENCES

[1] Cilium: Helping Linux Secure Microservices. https://www.cilium.io/. [2016-12-12].
[2] A Abhashkumar et al. P5: Policy-driven optimization of P4 pipeline *(SOSR'17).*
[3] O Hohlfeld et al. Demystifying the Performance of XDP BPF *(NetSoft 2019).*
[4] N Kang et al. Optimizing the "One Big Switch" Abstraction in SDN *(CoNEXT'13).*
[5] M Liu et al. IncBricks: Toward In-Network Computation with an In-Network Cache *(ASPLOS'17).*
[6] F Rath et al. SymPerf: Predicting Network Function Performance *(SIGCOMM Posters and Demos '17).*
[7] M Xhonneux et al. Leveraging eBPF for programmable network functions with IPv6 Segment Routing *(CoNEXT'18).*

NetPL '19, August 23, 2019, Beijing, China