ICAI 2023
12th International Conference on Applied Informatics
Eger, Hungary, 2023

P4 Specific Refactoring Steps[*]

Máté Tejfel, Dániel Lukács, Péter Hegyi

Faculty of Informatics
Eötvös Loránd University, ELTE
Budapest, Hungary
matej@inf.elte.hu, dlukacs@inf.elte.hu, immsrb@inf.elte.hu

Abstract

P4 [1] is a domain-specific programming language for programmable switches running inside next-generation computer networks. The language is designed to use the software-defined networking (SDN) paradigm [5] which separates the data plane and the control plane layers of the program. P4 focuses on the data plane, while we need some other tool to create the control plane. It facilitates the implementation of the concept of fully programmable networks making possible the development of programmable data planes.

This paper introduces certain tool-supported refactoring steps for P4 with two-fold objectives. On the one hand, we aim to assist developers to take full advantage of the programmability of P4, by providing standard refactoring services commonly found in IDEs of modern high-level languages. On the other hand, we want to enable P4 code optimizations that are aware of the unique make-up of this language.

The challenge in this task is that P4 has special domain-specific constructs that cannot be found in other languages and as such there is no existing methodology yet for refactoring these constructs. One of the most important part of the execution of a P4 program is the application of the match/action tables. The program can modify the incoming packets (namely the headers) by applying the tables. For a given program, determining the optimal table structure is a difficult task. It can often be the case that in a given hardware environment using fewer but larger

[*]The research has been supported by the project "Application Domain Specific Highly Reliable IT Solutions" implemented with the support of the NRDI Fund of Hungary, financed under the Thematic Excellence Programme TKP2020-NKA-06 (National Challenges Sub programme) funding scheme.
This research is in part supported by the project no. FK_21 138949, provided by the National Research, Development and Innovation Fund of Hungary.

Table 1. Vertical splitting

src_addr	*dst_addr*	action
1	1	*ipv4_forward*
1	2	*ipv4_forward*
1	3	*drop*
2	1	*drop*
2	2	*modify_dst*
3	1	*modify_dst*

(a) Table before vertical splitting

src_addr	action
1	case_1
2	case_2
3	case_3

(b) Vertical dispenser

executor_1	
dst_addr	action
1	*ipv4_forward*
2	*ipv4_forward*
3	*drop*

executor_2	
dst_addr	action
1	*drop*
2	*modify_dst*

executor_3	
dst_addr	action
1	*modify_dst*

(c) Executor tables

tables, while in another environment using more but smaller tables may yield better results. Therefore our implemented refactorings mainly focus on the modification of the match/action tables, namely horizontal and vertical splitting of tables, merging tables, changing the execution order of tables.

Table 1 illustrates vertical splitting. The parameter of the operation is a value set which determine the likely values of the first key part which can appear in the table. Using this set we can split the original table (1a) in such a way that first we just lookup the first key part in the determined set (1b) and then lookup the second key part in independent tables (1c).

The introduced refactorings were realized with P4Query [3], a static analysis framework for P4. However P4Query was originally designed with code discovery in mind rather than code transformation. As such, one of the issues a refactoring step has to address is keeping the database in a consistent state. Our current solution applies generic consistency checks after the execution of the refactoring steps for the internal graph to address this problem. Illustrating generality of our approach in addition to these steps, we have implemented some more general (not P4 specific) transformation steps (e.g. parameter renaming and magic number replacing) too. In the case of every refactoring we have also defined the prerequisites needed for the safe execution of the steps.

Considering related work P4 refactorings can be particularly useful for data-plane disaggregation, a problem recently addressed by Flightplan [6]. The objective here is to optimally segment P4 programs so that individual program segments can be assigned to different hardware resources. Another potential application for P4 refactorings can be the decomposition of large match/action tables containing significant amount of redundancy (due to dependencies between their fields) into smaller, irredundant tables using the approach introduced by Németh et al. [4]. On many targets decomposition leads to better efficiency, because smaller, simpler tables can be updated by the controller with less work, and because it is easier for compilers to find optimal representations for simpler tables. Our current approach to refactoring P4 using P4Query was mostly inspired by RefactorErl [2], a similar, but more established static analysis tool for Erlang, also developed at ELTE. A key idea in this tool is that persisting pre-calculated semantic information in a database can both simplify and speed up refactorings. This also enables incremental refactorings where syntactical changes automatically trigger semantic analysis.

In the future using the approach introduced by the paper we plan to implement further, more complex refactoring steps and to extend currently applied consistency checks with more specific verification methods.

References

[1] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, D. Walker: *P4: Programming Protocol-independent Packet Processors*, SIGCOMM Comput. Commun. Rev. 44.3 (July 2014), pp. 87–95, issn: 0146-4833, doi: 10.1145/2656877.2656890, url: http://doi.acm.org/10.1145/2656877.2656890.

[2] Z. Horváth, L. Lövei, T. Kozsik, R. Kitlei, M. Tóth, I. Bozó, R. Király: *Modeling semantic knowledge in Erlang for refactoring*, in: International Conference on Knowledge Engineering, Principles and Techniques (KEPT), Cluj-Napoca, Romania, July 2009, pp. 38–53.

[3] D. Lukács, G. Tóth, M. Tejfel: *P4Query: Static Analyser Framework for P4*, Annales Mathematicae et Informaticae. Liceum University Press, Eszterházy Károly University, Eger, Hungary. [**Submitted in 2022, under second round review**].

[4] F. Németh, M. Chiesa, G. Rétvári: *Normal Forms for Match-Action Programs*, in: Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies, CoNEXT '19, Orlando, Florida: Association for Computing Machinery, 2019, pp. 44–50, isbn: 9781450369985, doi: 10.1145/3359989.3365417, url: https://doi.org/10.1145/3359989.3365417.

[5] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, T. Turletti: *A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks*, IEEE Communications Surveys and Tutorials 16.3 (2014), pp. 1617–1634, doi: 10.1109/SURV.2014.012214.00180.

[6] N. Sultana, et al.: *Flightplan: Dataplane Disaggregation and Placement for P4 Programs*, in: 18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021, April 12-14, 2021, ed. by J. Mickens, R. Teixeira, USENIX Association, 2021, pp. 571–592, url: https://www.usenix.org/conference/nsdi21/presentation/sultana.