Towards Disaggregated P4 Pipelines with Information Exchange Minimization

Hiba Mallouhi
ELTE Eötvös Loránd University
Budapest, Hungary
hibamallouhi@inf.elte.hu

Sándor Laki
ELTE Eötvös Loránd University
Budapest, Hungary
lakis@inf.elte.hu

ABSTRACT

With the advent of programmable data plane hardware and the use of P4 as a high-level programming language, switches not only implement existing protocols but can also offload other tasks easily. However, hardware switches introduce several constraints on the number of computational stages, available SRAM/TCAM, ALUs, etc. In future use cases, such a device may potentially host multiple pipelines at the same time, resulting in additional constraints on the available resources for a single pipeline. The limitations of a single switch can however be handled by disaggregating the pipeline and deploying the resulted subpipelines at different switches in the network. In such approach, in addition to handling routing and packet classification, we also have to solve the information exchange between the depending subpipeline elements, adding extra headers transmitting metadata and variables needed for the continuation of the pipeline execution on another switch in the network. This paper explores the disaggregation of P4 pipelines with long-dependency sequences into subpipelines that minimizes the information exchange overhead and considers constraints on the maximum length of dependency chains in each subpipelines in a way to reach a fully automated process eventually.

CCS CONCEPTS

• **Networks** → **Programmable networks**;

KEYWORDS

P4, Pipeline Disaggregation, Dependency graph, Dependency sequence, Overhead minimization, Programmable networks.

ACM Reference Format:
Hiba Mallouhi and Sándor Laki. 2022. Towards Disaggregated P4 Pipelines with Information Exchange Minimization. In *CoNEXT Student Workshop 2022 (CoNEXT-SW '22), December 9, 2022, Roma, Italy*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3565477.3569156

1 INTRODUCTION

In the past few years, different methods emerged to optimize P4[2] pipelines or disaggregate them between various devices in the network. In Flightplan[5], they do the splitting of P4 program by manually adding segmentations to the different functions of the program while we focus on splitting by adding the constraints of a long-dependency and overhead minimization. In [3] they implement disaggregation by changing the physical layout and adding a crossbar and memory clusters, we argue that this will add extra cost and overhead that we want to minimize. Though these pioneering papers analyze and address several issues related to how pipeline disaggregation and optimization can be automated, they require manual steps in most cases, keeping lots of questions unsolved.

In this paper, we introduce our preliminary results towards an automated P4 pipeline disaggregation framework that will be able to handle multiple constraints and objectives, and generate the corresponding P4 programs implementing the disaggregated pipeline elements. A conceptual overview of the proposed method is depicted in Fig. 1. As an initial step, we assume the P4 program is represented as a program graph in **step 1**, which is used to generate a table dependency graph of edges (E) and vertices (V). As introduced in[4], we currently consider match and action dependencies only: 1) A match dependency occurs between two tables when the second table needs to match on a field or variable that was modified by the first table. 2) An action dependency occurs between two tables when the second table needs to modify a field or variable that was also modified by the first table. This means some tables in the pipeline cannot be executed before the tables they are dependent on, resulting in a constraint on the placement order of tables [3]. Each edge in the dependency graph is annotated with the variables and fields causing the dependency. Since a complex P4 program might have a long dependency sequence between its tables, we argue that disaggregating such pipeline into subpipelines will result in better utilization of the programmable data plane hardware or make possible the deployment of P4 programs that are too complex for a single device. We assume that this step can be done with tools like P4Query [1]. Our contribution is related to **step 2** in which we partition tables into two disjoint groups P_1 and P_2, applying Alg. 1. We assume there is a specific dependency limit for each switch as an abstract stage constraint that limits the longest dependency chain between tables inside each partition. A possible partitioning ensures that there is no edge in the dependency graph from any tables in P_2 to a table in P_1. Each partition should also meet the requirement on the dependency limit. The resulting overhead header to be transferred between the two subpipelines includes the variables and fields that belong to dependencies from P_1 to P_2 and are not stored in valid packet headers. In addition, we also transfer a branch point variable (e.g., 8 bit long) that marks the terminating position in the first subpipeline and is used in the second subpipeline to continue the pipeline execution from the correct state. Our disaggregation method aims to minimize the overhead header between the partitions. It calculates the overhead resulting

CoNEXT-SW '22, December 9, 2022, Roma, Italy

ACM ISBN 978-1-4503-9937-1/22/12…$15.00
https://doi.org/10.1145/3565477.3569156

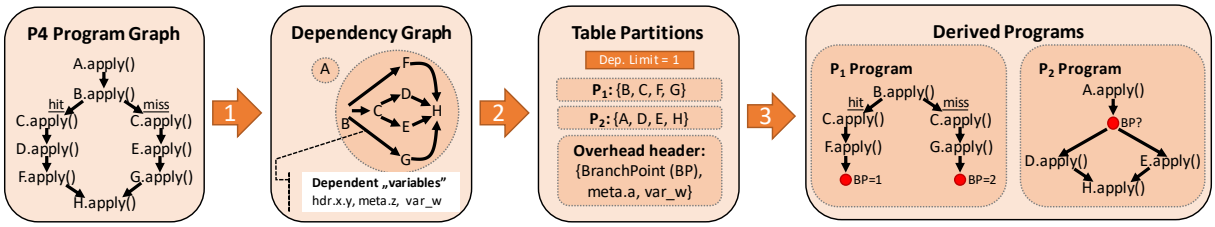

Figure 1: Proposed Disaggregation Method

from the variables transfer between the subpipelines and finds the minimum overhead exchange among the sets of possible partitions that satisfy the constraint on the dependency limit. This step may lead to multiple partitioning results with the same minimum overhead. Thus further objectives could be incorporated in the selection of the best partitioning. After having the partition, the proposed method derives two P4 programs representing the disaggregated subpipelines, shown as **step 3** in the figure. Step 3 is part of our future work. The proposed algorithm can be solved in polynomial time of $O(n^3)$. We note that in this paper we only focus on step 2 that is responsible for creating possible partitions that minimizes the transmission overhead between executing switches. We assume that Step 1 can be done by tool like P4Query [1], while step 3 is part of our future work.

Algorithm 1 Overhead minimization-based disaggregation

Table Set, $T = \{t_1, t_2, \ldots, t_n\}$
Dependency Graph, $G = (E, V)$ where $V = T$
Dependency-Variable Map, $Var(e) = \{a_1, \ldots\}$ $(e \in E)$
Per-Switch Dependency Limit, $L \geq 0$
$o_{min} \leftarrow \infty$; $OUT = \emptyset$
for each possible partitions (P_1, P_2) **do**
  if $\forall$path R_i in G_{P_i}: length$(R_i) \leq L$ $(i = 1, 2)$ **then**
    $o, v \leftarrow$ overhead(P_1, P_2, G, Var)
    if $o < o_{min}$ **then**
      $o_{min} = o$; $OUT = \{(P_1, P_2, v)\}$
    end if
    if $o = o_{min}$ **then**
      $OUT = OUT \cup \{(P_1, P_2, v)\}$
    end if
  end if
end for
return o_{min}, OUT

2 PRELIMINARY RESULTS

In this section, we evaluate our partitioning method (Alg. 1) on three P4 programs (L3dc.p4 and L2L3Complex.p4 taken from [4], and a toy program l2l3simpleMtag.p4). The details about the programs are summarized in Table 1. The listed parameters include the number of tables(n_T), the number of match-action-dependencies (n_mad), the number of variable in match-action-dependencies (n_mad_Var) and the length of the longest dependency chain LS.

P4 program	(n_T)	(n_mad)	(n_mad_Var)	LS
L3dc.p4	11	9	12	3
l2l3simpleMtag.p4	13	11	11	6
L2L3Complex.p4	24	27	34	8

Table 1: P4 programs and their parameters

Table 2 shows the partitioning results of our algorithm. For different dependency limits (L) we show two values (O, N) in each cell: the size of the minimum overhead header in bits (without the branch point variable) and the number of partitioning results with the same minimum overhead, resp. It is worth noting that for L3dc.p4 when $L = 2$, there are 6 sets with only 16 bits of overhead to be exchanged while l2l3simpleMtag.p4 has no possible sets for the same limit. l2l3simpleMtag.p4 has one bit of overhead to exchange for the four possible optimal cases when $L = 3$ and L2L3Complex.p4 has the 28 bits for the 3 sets of $L = 5$. There are also simple cases when $O = 0$, meaning that there is no dependency between the two partitions. These cases led to imbalanced table distribution between the two partitions.

P4 program	$L = 2$	$L = 3$	$L = 4$	$L = 5$
L3dc.p4	(16,6)	(0,6)	(0,6)	(0,6)
l2l3simpleMtag.p4	$\emptyset$	(1,4)	(0,8)	(0,17)
L2L3Complex.p4	$\emptyset$	$\emptyset$	$\emptyset$	(28,3)

Table 2: (O, N): minimum overhead and number of optimal partitioning cases

3 FUTURE DIRECTIONS

We present our preliminary results towards a framework that aims to automate P4 program disaggregation. Our future work includes: 1) Extending the scope of disaggregation to multiple subpipelines considering a more complex topology such as a tree-shaped network as we currently explore a linear pipeline only. 2) Introducing additional pipeline resources (e.g., SRAM and TCAM) needed to enable disaggregation with "branch-and-resume" between multiple pipelines and other hardware-specific constraints.

ACKNOWLEDGMENT

Authors thank the support of the National Research, Development and Innovation Office – NKFIH, FK 138949.

REFERENCES

[1] 2022. P4Query tool. https://github.com/P4ELTE/P4Query. (2022). Accessed: 2022-06-01.

[2] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 87–95.

[3] Sharad Chole, Andy Fingerhut, Sha Ma, Anirudh Sivaraman, Shay Vargaftik, Alon Berger, Gal Mendelson, Mohammad Alizadeh, Shang-Tse Chuang, Isaac Keslassy, et al. 2017. drmt: Disaggregated programmable switching. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 1–14.

[4] Lavanya Jose, Lisa Yan, George Varghese, and Nick McKeown. 2015. Compiling packet programs to reconfigurable switches. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. 103–115.

[5] Nik Sultana, John Sonchack, Hans Giesen, Isaac Pedisich, Zhaoyang Han, Nishanth Shyamkumar, Shivani Burad, André DeHon, and Boon Thau Loo. 2021. Flightplan: Dataplane disaggregation and placement for p4 programs. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 571–592.