Compiling Packet Programs to Reconfigurable Switches: Theory and Algorithms

Balázs Vass
Budapest University of Technology and Economics
Budapest, Hungary
vb@tmit.bme.hu

Erika Bérczi-Kovács
Eötvös Loránd University (ELTE) and MTA-ELTE
Egerváry Research Group on Combinatorial Optimization
Budapest, Hungary
koverika@cs.elte.hu

Costin Raiciu
University Politehnica of Bucharest
Bucharest, Romania
costin.raiciu@cs.pub.ro

Gábor Rétvári
Budapest University of Technology and Economics
Budapest, Hungary
retvari@tmit.bme.hu

ABSTRACT

A critical step in P4 compilation is finding a feasible and efficient mapping of the high-level P4 source code constructs to the physical resources exposed by the underlying hardware, while meeting data and control flow dependencies in the program. In this paper, we take a new look at the algorithmic aspects of this problem, with the motivation to understand the fundamental theoretical limits and obtain better P4 pipeline embeddings, and to speed up practical P4 compilation times. We report mixed results: we find that P4 compilation is computationally intractable even in a severely relaxed formulation, and there is no hope for a tractable approximation of arbitrary precision, while the good news is that, despite its inherent complexity, P4 compilation is approximable in quasi-linear time with a small constant bound even after removing most of the relaxations from the model.

CCS CONCEPTS

• **Networks** → **Network algorithms**; • **Hardware** → **Networking hardware**.

KEYWORDS

reconfigurable switches, packet programs, pipeline embedding, P4 compilation, algorithmic complexity, constant approximations

ACM Reference Format:
Balázs Vass, Erika Bérczi-Kovács, Costin Raiciu, and Gábor Rétvári. 2020. Compiling Packet Programs to Reconfigurable Switches: Theory and Algorithms. In *3rd P4 Workshop in Europe (EuroP4'20), December 1, 2020, Barcelona, Spain.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3426744.3431332

EuroP4'20, December 1, 2020, Barcelona, Spain

ACM ISBN 978-1-4503-8181-9/20/12…$15.00
https://doi.org/10.1145/3426744.3431332

1 INTRODUCTION

Future computing applications critically depend on more efficient, reliable, flexible, and observable networks [13]. Accordingly, programming reconfigurable switch pipelines using a high-level domain-specific language like P4 [4] is increasingly being adopted in diverse application areas, like large-scale disaggregation, in-network computation [6], telemetry [12], load-balancing [11, 14], etc. With applications booming, we witness dataplane programs growing in complexity, including more and larger match-action tables, diverse header parse graphs, table–action dependency relationships, and match and action types [2]. At the same time, new generations of programmable switch ASICs [3] feature even more dataplane resources and pipeline stages.

Dataplane programming adopts a top-down approach: the required behavior of the network is described in a declarative P4 program, which is then mapped to the underlying hardware by a *P4 compiler*. The compiler must analyze the P4 program and, given an abstract model of the hardware target including limits on the available memory space, width, and types, the number of processing stages, and the supported level of concurrency at each stage, find the best encoding of the match-action tables into the target switch pipeline so that control and data dependencies in the P4 program are reproduced in a semantically correct way. Here, the "best" encoding may be such that it minimizes the number of required stages or the associated power consumption. We call this task the *pipeline embedding* problem.

The seminal paper [10] sets the stage for P4 program compilation, proposing an abstract model to describe the resource requirements and data-/control-dependencies of P4 programs as well as the switch dataplane resources, and presents various algorithms for pipeline embedding. In particular, an Integer Linear Program (ILP) is proposed to obtain an optimal solution, but with possibly exponential running time and/or memory footprint, and a greedy heuristics is also presented to get quick embeddings but with a possibly unlimited optimality gap (margin from the optimal solution).

Unfortunately, the most important algorithmic questions related to pipeline embedding have remained open since this seminal paper. This is becoming increasingly troubling, with the trend of P4 programs getting ever larger and the underlying switch ASICs getting more complex. Accordingly, P4 compilation times may easily grow

Pipeline	#nodes	#arcs	ILP runtimes in [sec] for different gaps					Greedy runtime	Gap (greedy #stages to opt.)
			50%	30%	20%	10%	0%		
L3DC	11	13	1.29	1.32	1.34	1.35	1.30	0.055	14.2%
L2L3 simple	13	16	4.06	7.78	7.95	8.74	10.3	0.153	8.3%
L2L3 complex	24	35	293	703	2658	4315	4931	0.200	14.8%

Table 1: Running time of the ILP and a greedy algorithm of [10] for pipeline embedding on a commodity quad-core computer with CPU @ 2.3 GHz and 8 GB RAM. The ILP was run using parallel B&C on 4 threads. While the greedy approach scales with the network size, ILP runtimes expload.

beyond practical: Table 1 reports the typical pipeline embedding times on some common-case P4 programs obtained using the framework in [1, 10]. Indeed, finding the exact optimum may take more than an hour for the ILP even on a moderate size P4 program of 24 match-action tables and 35 control flow dependencies. Anecdotal evidence exists that it often takes several hours for a larger P4 program to be compiled with commercial P4 SDKs and ILP solvers, and sometimes even finding a feasible pipeline embedding is already a huge computational challenge. This makes P4 (re)compilation a painful and cumbersome process, complicating debugging, and wasting programmer time. At the same time, the greedy heuristics, which is guaranteed to run in polynomial time, may produce low-quality embeddings, and it may easily conclude that there is no feasible embedding even when there is one.

In this paper, *we take the first steps toward obtaining a comprehensive algorithmic landscape of P4 pipeline embedding*. To the best of our knowledge, ours is the first principled approach to this end. We take off from a heavily simplified model of a programmable switch target (the INF-CAP model) on which pipeline embedding maps to a well-known combinatorial optimization problem that can be solved to optimality in linear time. Then, *we re-introduce additional degrees of complexity* into the barebones model to obtain increasingly more realistic and restrictive, P4 pipeline models, and *we give a comprehensive characterization of the respective computational complexity and approximation bounds* and cast several open problems. We stress that our main contribution is *not* suggesting new tricks to improve P4 compilation/embedding efficiency, but rather to characterize the algorithmic aspects of *existing* techniques proposed in, e.g., [10].

Our results are summarized in Table 2. As the first practical application, we find that the greedy pipeline embedding heuristics proposed in [10] and implemented in [1] already provides an approximation with a small constant gap in several models, presenting an appealing choice when resource requirements are not that stringent.

2 P4 PIPELINE EMBEDDING

In this section, we build a sequence of increasingly complex models to characterize the resource requirement of realizing P4 programs. For each model, we analyze the computational complexity of the particular incarnation of the P4 pipeline embedding problem and, using classical results in combinatorial optimization, we derive the corresponding inapproximability (bad news) and approximability (good news) bounds. In the main part of the paper, we focus on the claims and present only simple sketches of the proofs; the details are relegated to the Appendix.

2.1 INF-CAP: Mapping Concurrency

The real complexity of P4 pipeline embedding stems from the difficulty of assigning the elements of the match-action logic to the physical resources of the switch pipeline so that the result is semantically correct dataplane behavior that matches the intent of the programmer. This difficulty, on the one hand, arises from the fact that there are *various control-flow dependencies*, implicit or explicit, in the P4 program that requires properly sequencing the match-action stages in the switch pipeline; for instance, a packet header field cannot be processed until a previous stage has finished modifying it. On the other hand, dataplane resources are inherently *restricted in size and the type of operations supported*; e.g., a TCAM can store only a limited number of ternary match entries and only of limited width. In this section, we concentrate on the first constituent of this complexity only, and we relax the second: the task is to correctly represent data- and control dependencies in a switch pipeline of infinite resources.

In this model, one would naively try to map the entire P4 match-action logic to the first stage (any large program fits into unlimited memory after all), obtaining a massively concurrent embedding with the smallest theoretically possible processing latency. This, however, most probably would result in an incorrect embedding due to the inherent control-flow dependencies in the P4 program: e.g., whenever table A modifies a field that table B matches, table A cannot be assigned to the same stage as table B (match-dependency), if both tables modify the same field then the one that is applied last must be assigned to a later stage (action-dependency), etc.; see [10] for the details.

From an algorithmic standpoint, such control-flow dependencies can be represented using a *Table Dependency Graph (TDG)*: given a P4 program, the corresponding TDG $T = (V(T), A(T))$ is a Directed

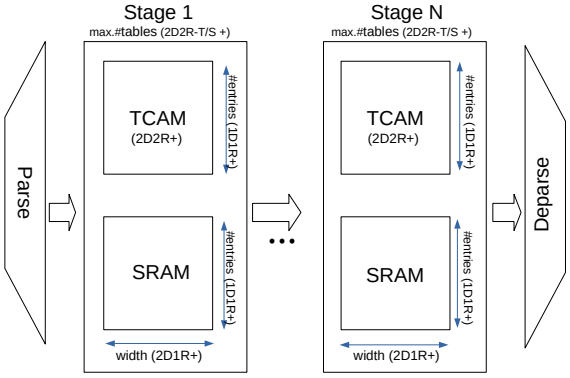


Figure 1: Switch pipeline models.

Model name	**INF-CAP**	**1D1R**	**1D1R-*hsplit***	**2D1R**	**2D2R**	**2D2R-T/S**	**2D2R-PISA**
New feature on top of the previous model	(mapping concurrency due to dependency)	1D capacity/ demands	*hsplit* (table entries split between stages)	2D capacity/ demands	2 kinds of resources per stage	limited number of tables per stage	crossbar constraints, word packing, etc.
Complexity	P	NPC	NPC	NPC	NPC	NPC	NPC
						strongly NP-hard	
Bad news: (unless P=NP,) Inapproximable better than ...	OPT	3/2∗OPT	5/4∗OPT	5/4∗OPT	5/4∗OPT	?	?
Good news: Constant-approximable in...	OPT	3∗OPT	2∗OPT	3∗OPT	(5 to 8)∗OPT (*)	(6 to 9)∗OPT (*)	?

Table 2: Overview of the main results. Bad news: the Pipeline Embedding Problem (PEP) is NP-Complete even with simple precedence and resource constraints. Good news: the PEP is constant approximable in quasi-linear time even after removing most relaxations from the model. (*) means the lower bound holds in certain natural conditions explained in Sec. 2.5.

Acyclic Graph (DAG) of $n(T) = |V(T)|$ vertices representing logical match-action tables (MATs), and $m(T) = |A(T)|$ arcs representing the different types of dependencies (match, action, etc.) that exist between the MATs. The packet processing pipeline can be modeled as a directed path with nodes $s_1, s_2, \ldots$ representing the pipeline stages, so that the arcs (s_i, s_{i+1}) encode the succession of stage s_i and s_{i+1} in the pipeline. To simplify the developments, we assume that the switch has infinitely many stages, so that the objective is to minimize the number of stages used by the embedding.

As the simplest formulation of the P4 pipeline embedding problem (INF-CAP), below we require that the MATs $V(T)$ be embedded in the switch pipeline such that the arcs of the TDG are "forward"; i.e. for every $(l_a, l_b) \in A(T)$, table l_a is mapped to a stage s_i and l_b is mapped to a stage s_j with $i < j$. However, we assume that all stages are of unlimited processing capability (i.e., can perform all types of matches and actions) and infinite size and "width" (resource limits will be introduced in the next sections).

Claim 1. *Pipeline embedding under the INF-CAP model can be solved to optimality in linear time:* $O\big(n(T) + m(T)\big)$.

The proof is fairly trivial: under INF-CAP one can obtain a correct embedding using topological sorting in polynomial time (in fact, linear) as follows. Add a hypothetical MAT L to the TDG with arcs (L, l_i) to all source nodes of the TDG, and for each j embed each MAT $l_k \in V(T)$ having a longest directed path of length j from L to stage j. It is easy to see that this algorithm returns a valid TDG embedding with the minimal theoretically possible stages (the length of the longest directed path in the TDG).

In hindsight, when the task is only to encode the control-flow dependencies but there are no resource limits, then the P4 pipeline embedding problem is "easy", which even a trivial greedy sorting heuristic (quite similar to the one proposed in [10]) solves to optimality rapidly. This finding is not particularly surprising: the types of control-flow dependencies in a P4 program are very similar to the dependencies occurring in general programming languages (read-after-write, write-after-write, etc.), and basically any compiler can routinely analyze (and optimize) such dependencies at large scale.

2.2 1D1R: Adding Simple Resource Constraints

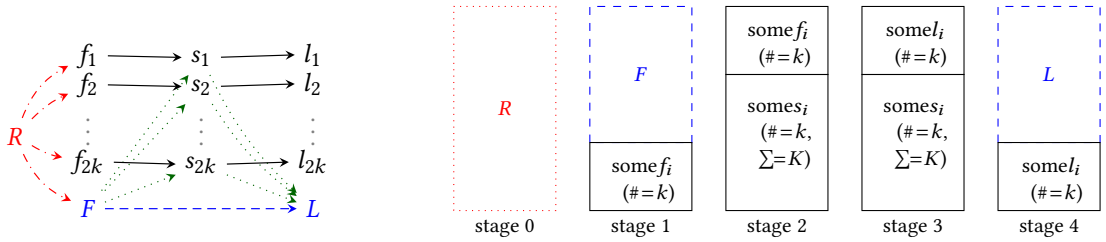
Easily, INF-CAP yields an embedding with the maximum possible concurrency. Unfortunately, this model is not too realistic since it relaxes all types of resource constraints.

In our second model called 1D1R, we assume that *each stage can store only a limited number of table entries*; i.e., each MAT has a one-dimensional ("1D") memory demand and each stage has a 1D memory constraint; for simplicity, we assume that all stages have the same capacity. For now, we presume that there is only a single type of memory resource ("1R") in the switch (TCAM or SRAM) that can perform all match and action operations required by the P4 program. Our main result for 1D1R is as follows.

Claim 2. *Pipeline embedding under the 1D1R model is NP-complete. Bad news: the optimal number of stages cannot be approximated better than* 3/2 *unless* $P = NP$. *Good news: the optimum can be* 3*-approximated in quasi-linear time*[1]*:* $O\big(n(T) \log n(T) + m(T)\big)$.

Intuitively speaking, the hardness result appears since 1D1R contains NP-hard problems as a special case, like `EQUAL CARDINALITY PARTITION` or `PARTITION`; see Fig. 2 for a sketch of the construction required to obtain the intractability result. Inapproximability, furthermore, can be proved by a similar construction: denoting the number of stages with K, consider M instances of the `PARTITION` problem, where the sum of the numbers (representing the maximum number of entries in tables) is $2K$. From every table in the k-th instance, there is an arc to every table in the $(k+1)$-th instance, this way, the instances are placed in separate stages. By construction, if an instance has a solution, then it can be embedded in 2 stages, otherwise it needs at least 3 stages. This means that if all the instances have a solution, then $2M$ stages are needed, but since the `PARTITION` is NP-complete [9] one cannot give a solution that uses less than $3M$ stages unless P=NP. Finally, 3-approximability can be stated constructively, using an algorithm similar to the one that in the previous section solving INF-CAP (see Claim 7 in Appendx C).

[1] Henceforth, for each table, we only store its starting stage and position, and its end stage and position, resp. The actual embedding phase needs some more time that scales with both the stage sizes and the optimal stage number.



(a) TDG. $\sum_{i\in\{1,\dots 2k\}} e(s_i) = 2K$, $e(f_i) \equiv e(l_i) \equiv 1$, $e(F) = e(L) = K$, $e(R) \leq K + k$

(b) The only possible scheme of a feasible embedding in case of stages having maximal capacities of $K + k$ entries

Figure 2: Sketch of proof of NP-completeness of the Pipeline Embedding Problem (PEP) variants 1D1R, 1D1R-*hsplit*, 2D1R, 2D2R. A valid embedding is a polynomial witness for the decision versions. The pipelines can be embedded in stages nr. $0, \dots, 4$ exactly if the `EQUAL CARDINALITY PARTITION` problem has a solution over $\{s_1, \dots, s_{2k}\}$, which is NP-hard [9] to decide. Note that due to construction, the *hsplit* cannot be used in a valid embedding. Details in Appendix A.

The bottom line is that pipeline embedding, and hence P4 compilation, becomes intractable and inapproximable with PTAS at the instance we consider, however simple resource limits. On the other hand, 3-approximability under 1D1R is particularly good news: one can quickly develop a heuristic embedding using a variant of the greedy heuristic in [10] and can be *provably* safe that the resultant embedding will use at most 3 times the number of stages of the optimal embedding.

2.3 1D1R-*hsplit*: Table Splitting

In the 1D1R model, we require that each MAT is assigned to a unique pipeline stage as a whole; the intractability of the related pipeline embedding problem can then be viewed as a consequence of the intricate combinatorial structure such *packing* constraints typically introduce. However, it is not unusual for some MATs to exceed the storage capacity of a single pipeline stage; e.g., a large IP routing table may not fit into a single TCAM bank. To address this challenge, [10] presents a technique to split MATs across multiple stages: one stores as many entries in the first stage as possible and, should a match be found, jump to the next processing stage; otherwise, matching falls back to the second (split) portion of the MAT through a default "catch-all" rule. We call this operation a *horizontal split* (or *hsplit*). Intuitively, permitting *hsplit* renders the embedding problem a bit easier, in that it allows a MAT to be mapped "fractionally", splitting it into multiple MATs placed consecutively in the pipeline, which relaxes the packing constraints from 1D1R. Next, we ask whether this relaxation renders the pipeline embedding problem any more approachable computationally.

Formally, the 1D1R-*hsplit* model we consider below is a version of 1D1R where the compiler is allowed to perform horizontal splits on MATs. Here, *hsplit* is a specific transformation of the input TDG whereby any MAT $U \in V(T)$ is substituted with two MATs $(A, B) = hsplit(U)$ so that *(i)* U is replaced with A and B, *(ii)* both A and B inherit all arcs of U, *(iii)* a new arc (A, B) is created, and finally *(iv)* the sizes of A and B sum up to the size of U. Note that *hsplit* can be applied recursively to split a MAT into more than 2 tables. Our complexity results on 1D1R-*hsplit* are as follows:

Claim 3. *Pipeline embedding under the 1D1R-hsplit model is NP-complete. Bad news: the optimal number of stages cannot be approximated better than 5/4 unless $P = NP$. Good news: the optimum can be 2-approximated in linear time: $O(n(T) + m(T))$.*

Algorithm 1: 1D1R-*hsplit* First Fit by Level

Input: TDG $T(V(T), A(T))$; stage size c_s; node sizes $e(v)$
begin
1 $[v_1, v_2, \dots, v_n] :=$ `TopologicalOrdering`$(T(V(T), A(T)))$
2 **for** $i = 1, \dots, n$ **do**
3 $\mathcal{L}(i) :=$ empty list
4 **if** v_i *does not have any in-arc* **then**
$l(v_i) := 1$
5 **else**
$l(v_i) := \max\{l(v_j) | (v_j, v_i) \in A\} + 1$
6 append v_i to $\mathcal{L}(l(v_i))$
7 $E(0) := 0$, $i := 1$
8 **while** $\mathcal{L}(i) \neq \emptyset$ **do**
9 $E(i) := E(i-1) + \lceil (\sum_{v\in\mathcal{L}(i)} e(v))/c_s \rceil$
10 **for** $v \in \mathcal{L}(i)$ **do**
11 Reserve space for v continuously (using *hsplit*) in the next free spaces in stages $S[E(i-1)+1], \dots, S[E(i)]$
12 $i := i + 1$
13 **return** *Embedding*

Below, we sketch an algorithm to constructively obtain the 2-approximation on 1D1R-*hsplit* (see Alg. 1). This algorithm is motivated by the design of the greedy heuristics of [10]; accordingly, the approximability result can be applied to the generic greedy scheme under 1D1R-*hsplit* with minor modifications.

Theorem 1. *The "First Fit by Level" (or simply, FFL) heuristic (Alg. 1) gives a 2-approximation for the pipeline embedding problem under 1D1R-hsplit in linear time: $O(n(T) + m(T))$.*

Proof: *Correctness:* For any $u, v \in V(T)$, (u, v) being a dependency arc means $l(v) > l(u)$ in the output, where $l(w)$ denotes the stage table w is embedded. Thus, Alg. 1 returns a correct embedding. *2-approximation:* The minimum number of stages needed for embedding the TDG (OPT) is at least the number h of used stages which are not full after running the FFL since h is at most the length of the longest dependency chain in the TDG. On the other hand, OPT is at least the number of stages fully utilized by FFL. Consequently, FFL uses at most $2 * \text{OPT}$ stages. *Complexity:* At line 1 we return a topological ordering, this can be done in $O(n(T) + m(T))$. By pre-storing the in-arcs for each node (in $O(n(T) + m(T))$), executing lines 2–6 also takes $O(n(T) + m(T))$ time. Finally, at lines 8–11, $O(n(T) + m(T))$ operations are done again. □

The essence is that the additional degree of freedom introduced by *hsplit* does not eliminate intractability, which explains why the

ILP fails to find an optimal embedding in reasonable time. At the same time, 1D1R-*hsplit* admits slightly more favorable approximability guarantees: we found that under 1D1R-*hsplit* one can quickly find an embedding that uses at most 2 times the number of stages in an optimal embedding. Put it another way, the output of Alg. 1 can be used as a good initial primal feasible solution to bootstrap the ILP, which can then at most halve the number of the stages used. Meanwhile, adopting *hsplit* also reduces our best inapproximability bound (from 3/2 to 5/4).

2.4 2D1R: Two-dimensional Resources

Up to this point, we have assumed that the stages can store a predefined number of table entries, regardless of the width of these entries. In reality, MATs have (at least) two size dimensions (number of entries and header field match-width), and similarly, pipeline memory has two-dimensional capacities. Obviously, pipeline embedding should respect both size dimensions: given the number of entries n_u and the width l_u of each MAT $u \in V(T)$, neither the number of entries nor the total width of the MATs placed to any stage can exceed the 2D capacity of that stage. Below, we consider the simplest form of this model called 2D1R that is akin to 2D geometric bin-packing [5]; we note that real ASICs typically exhibit several additional subtle complexities, e.g., memory can be allocated only in discrete blocks, MATs cannot be arbitrarily placed side-by-side, etc.; we ignore these here for brevity. Further note that *hsplit* is still permitted.

As shown next, this second problem dimension makes pipeline embedding slightly more complex:

Claim 4. *Pipeline embedding under the 2D1R model is NP-complete. Bad news: the optimal number of stages cannot be approximated better than 5/4 unless $P = NP$. Good news: the optimum can be 3-approximated in quasi-linear time: $O\big(n(T)\log n(T) + m(T)\big)$.*

Here, only 3-approximability needs more explanation: we again motivate this result with a (sketch of a) constructive proof. The approximation algorithm for 2D1R is quite similar to Alg. 1 we used for 1D1R-*hsplit*, the only difference is in the packing of the levels. Initially, inside each level $L(i)$, we sort nodes into a descending order of width. Without loss of generality, scale the widths such that each stage is of width 1. First, MATs of width $\geq 1/2$ are packed (in one column), then MATs of width in $[1/3, 1/2)$ are packed in two columns, etc. Generally, after packing of a set of MATs of width $[1/k, 1/k-1)$ is finished, a new set of MATs $V_{w,k}$ of width $[1/(k+1), 1/k)$ is packed in k columns such that the length of the columns is almost equal; and if the last row of $V_{w,(k-1)}$ remains unfilled then we start packing $V_{w,k}$ by assigning some of its elements to the remaining space of the row. One can show that with this modification Alg. 1 returns a valid pipeline embedding in polynomial time that 3-approximates the number of the stages in the optimal solution under 2D1R; roughly speaking, the idea is that there are at most $\leq$OPT stages that are not at least halfway full after Alg. 1 terminates (see Claim 8 in Appendix C for details).

2.5 2D2R: Both SRAMs/TCAMs Available

In most switch ASICs, there are multiple types of memory, each optimized for different purposes [3]: TCAMs excel at prefix and wildcard matches but come at a considerable power budget and cost, while SRAMs are ideal for performing exact or range matches or to store action code. Confusingly, some MATs may be assigned to any type of memory (e.g., a TCAM can also perform exact matches). We call the pipeline model with 2 resource types as 2D2R.

One easily concludes that, being a stronger model than 2D1R, 2D2R is also NP-complete and inapproximable below 5/4 (unless P=NP). On the positive side, in Appendix C, we show a modification of Alg. 1 that attains an 8-approximation under 2D2R. Note that the width w_T of TCAMs might be different from the width w_S of SRAMs (that is scaled to be 1), and if $w_T \geq w_S$, our algorithm is a 5-approximation. In the modified algorithm, we still embed levels $\mathcal{L}_i$, $i \in 1, 2, \ldots$ separately. For a level $\mathcal{L}_i$, 1) we embed those MATs that can be mapped only to TCAMs, 2) if the width of SRAMs is greater than the width of TCAMs (i.e., $w_S > w_T$), we embed those MATs that can be mapped only to SRAMs (due to their width being $> w_T$), 3) we embed the remaining MATs to the remaining memory of these stages, and, if needed, we open up additional stages[2]. In all the previous three phases, the embedding is done following the steps of the 2D1R version of Alg. 1, starting from the first stage assigned to the level. The proof of 8- (and 5-) approximation can be found in Appendix C as proof of Claim 9.

Hence, our results establish appealing approximation bounds of greedy heuristics under 2D2R as well:

Claim 5. *Pipeline embedding under the 2D2R model is NP-complete. Bad news: the optimal number of stages cannot be approximated better than 5/4 unless $P = NP$. Good news: the optimum can be 8-approximated in quasi-linear time: $O\big(n(T)\log n(T) + m(T)\big)$. If $w_T \geq w_S$, it can be 5-approximated in the same complexity.*

2.6 2D2R-T/S: Constrained Number of Tables per Stage

In recent programmable switch ASICs [3] the number of MATs that can be assigned to a single stage is limited by the capacity of the memory chips and the crossbars connecting the stages. Below we (re-)introduce this constraint to obtain the 2D2R-T/S model, and we find that with this additional complexity, the problem becomes strongly NP-hard.

Claim 6. *Pipeline embedding under 2D2R-T/S is NP-complete and NP-hard in the strong sense. Good news: the optimum can be 9-approximated in quasi-linear time: $O(n(T)\log n(T)+m(T))$. If $w_T \geq w_S$, it can be 6-approximated in the same complexity.*

Proof: *Complexity:* Consider the decision version of pipeline embedding, where the question is whether the TDG can be embedded into a given number of stages. Let all the stages and MATs have equal width, thus the second size dimension can be ignored. Let all the MATs be assignable to any of the resources, thus we can characterize the size of each stage with only the number of table entries it can accommodate. Let the pipeline have m stages of size $B \in \mathbb{Z}^+$, each stage capable of storing at most 3 tables. Let the TDG be embedded consist of $3m$ MATs of size in $\{B/4, \ldots, B/2\}$, without any TDG dependencies. The sizes of the nodes add up to mB. Then, the decision whether the TDG can be embedded into the pipeline is equivalent to the `3-PARTITION` problem, which is known to be

[2]Note that, for each level, in a particular setting discussed in the Appendix we need a fourth phase of our algorithm that affects two rows, possibly emptying a stage.

strongly NP-hard [9, SP15]. A valid embedding is a polynomial witness, completing the proof of NP-completeness.

9- *(or even 6−) approximation:* We use the greedy heuristic scheme of version 2D2R, with the modification that wherever the number of MATs assigned to the current stage would exceed the allowed maximum c we start a new stage. Then, one can show that there is only OPT more stages needed for embedding under 2D2R-T/S than under 2D2R, since the number of stages hosting exactly c MATs is $\leq$OPT. Consult proof of Claim 10 in Appendix C for the details. □

2.7 2D2R-PISA: Fully-fledged PISA Model

The constraints incurred by a real switch go beyond our simplified models; see [1, 10]. Incorporating all these operational constraints into a (hypothetical) 2D2R-PISA model, we see that solving pipeline embedding over 2D2R-PISA is at least as difficult as on 2D2R-T/S as it contains that as a special case (strongly NP-hard). Designing an efficient approximation for this model would exceed the limits of this paper.

3 CONCLUSION AND FUTURE WORK

P4 pipeline embedding sits at the core of programmable dataplane devices, as it is a crucial step for deploying P4 programs on real targets. Given the stringent resource limits of such targets and the size of P4 programs that must be deployed, achieving optimal solutions to pipeline embedding is really important. In this paper, we have taken a first look at the theoretical aspects of this problem, characterizing its complexity and providing bounds for different models. Our results will serve as a tool to judge the performance of future P4 compilers, as well as guidance to build better compilers.

Higher level languages like μP4 [15] and Lyra [7] have been recently proposed to raise the abstraction level for programmers and make it easier to write portable programs that can run on multiple platforms. Both these works have to solve the pipeline embedding problem as a final step before deployment, and they currently use greedy approaches.

Chipmunk [8] takes a different approach: it formulates pipeline embedding as a program synthesis problem and uses Sketch to generate programs that fit the target constraints. To tame complexity, Chipmunk breaks up the program into smaller pieces and compiles each part independently; this makes the program tractable in the examples shown, but can result in globally sub-optimal outcome. While Chipmunk outperforms Domino, a greedy compiler for pipeline embedding, it does not claim optimality; in our future work, we intend to analyze the optimality gap between Chipmunk and other greedy approaches. Furthermore, we aim to tighten our lower and upper bounds for the different pipeline embedding variants and provide a clear picture of how far existing greedy approaches compare to the optimal.

ACKNOWLEDGEMENTS

This research is in part supported by the project no. 129589, 135606, FK_20 135074, FK 132524, K 124171, and K 128062, provided by the National Research, Development and Innovation Fund of Hungary. G.R. is with the MTA-BME Information Systems Research Group, the MTA-BME Momentum Network Softwarization Research Group, and Ericsson Research, Hungary. Costin Raiciu was partly funded by CORNET H2020, a research grant of European Research Council (no. 758815).

REFERENCES

[1] Bitbucket code repository of [10]. https://bitbucket.org/lavanyaj/switch-compiler/src/master/. Accessed: 2020-08-19.
[2] Switch.p4. https://github.com/p4lang/switch/blob/master/p4src/switch.p4. Accessed: 2020-08-19.
[3] Barefoot. Intel/Barefoot Tofino 2: Product Brief. https://www.barefootnetworks.com/products/brief-tofino-2. Accessed: 2020-09.
[4] Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM CCR 44*, 3 (2014), 87–95.
[5] Christensen, H. I., Khan, A., Pokutta, S., and Tetali, P. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review 24* (2017), 63–79.
[6] Dang, H. T., Canini, M., Pedone, F., and Soulé, R. Paxos made switch-y. vol. 55, ACM SIGCOMM, pp. 19–24.
[7] Gao, J., Zhai, E., Liu, H. H., Miao, R., Zhou, Y., Tian, B., Sun, C., Cai, D., Zhang, M., and Yu, M. Lyra: A cross-platform language and compiler for data plane programming on heterogeneous ASICs. In *ACM SIGCOMM* (2020), p. 435–450.
[8] Gao, X., Kim, T., Wong, M. D., Raghunathan, D., Varma, A. K., Kannan, P. G., Sivaraman, A., Narayana, S., and Gupta, A. Switch code generation using program synthesis. In *ACM SIGCOMM* (2020), p. 44–61.
[9] Garey, M. R., and Johnson, D. S. Computers and intractability: A guide to the theory of NP-completeness.
[10] Jose, L., Yan, L., Varghese, G., and McKeown, N. Compiling packet programs to reconfigurable switches. In *USENIX NSDI* (2015), pp. 103–115.
[11] Katta, N., Hira, M., Kim, C., Sivaraman, A., and Rexford, J. Hula: Scalable load balancing using programmable data planes. In *ACM SOSR* (2016), pp. 1–12.
[12] Kim, C., Sivaraman, A., Katta, N. P., Bas, A., Dixit, A., and Wobker, L. J. In-band network telemetry via programmable dataplanes.
[13] McKeown, N. Programmable forwarding planes are here to stay. In *ACM SIGCOMM NetPL* (2017).
[14] Miao, R., Zeng, H., Kim, C., Lee, J., and Yu, M. Silkroad: Making stateful Layer-4 load balancing fast and cheap using switching ASICs. In *ACM SIGCOMM* (2017), pp. 15–28.
[15] Soni, H., Rifai, M., Kumar, P., Doenges, R., and Foster, N. Composing dataplane programs with μP4. In *ACM SIGCOMM* (2020), p. 329–343.

APPENDIX

A NP-COMPLETENESS OF PEP VERSIONS 1D1R, 1D1R-*HSPLIT*, 2D1R, 2D2R

By definition, the task in Pipeline Embedding Problem (PEP) versions in this paper is to find an embedding of the TDG that uses the least stages possible. In other words, we are investigating the *minimization (optimization)* version of the problems. The *decision* version of these problems on the other hand ask if a given number of k stages is enough to embed a given TDG (or alternatively, if a TDG can be mapped on a given switch architecture). We can see that the NP-hardness of the decision version induces the NP-hardness of the minimization version (by the hardness of deciding whether a candidate number *min* of stages is the minimal possible). In the followings, we will prove the NP-hardness of different PEP variants through the NP-hardness of their decision versions.

Lemma 1. *PEP versions 1D1R, 1D1R-hsplit, 2D1R, and 2D2R are in NPC.*

Proof: Firstly, the decision versions of the problems are in NP, since given a candidate embedding, one can easily check its validity in polynomial time. This implies that the minimization versions are also in NP. We will prove the NP-hardness of the decision versions of the problems, which implies the NP-hardness of the minimization version. Let each table in TDG T have only exact matches. For this part of the proof, let us assume that *hsplit* is forbidden.

First, let us concentrate on the black-and-solid parts of Fig. 2: we have tables $f_1, \ldots, f_{2k}$, $s_1, \ldots, s_{2k}$, and $l_1, \ldots, l_{2k}$ to embed, and stages 1, 2, 3, and 4 with number of maximum entries of k,

$K + k$, $K + k$ and k, respectively. Each $\{(f_i, s_i), (s_i, l_i)\} \subseteq A(T)$. Since the number of maximum table entries are $e(f_i) \equiv e(l_i) \equiv 1$, and $\sum_{i \in \{1,\dots 2k\}} e(s_i) = 2K$, due to the stage sizes, the only way these tables can fit in the stages is if there exists a set of indexes $I \subset \{1, \dots, 2k\}$, $|I| = k$ such that for each $i \in I$, tables f_i are assigned to stage 1, s_i are in stage 2, l_i are in stage 3, while for $j \in \{1, \dots, 2k\}$, f_j assigned to stage 2, s_j are in stage 3, l_i are in stage 4, and $\sum_{i \in I} e(s_i) = K$. One can see that this kind of valid embedding exists exactly if there exists such an I ($|I| = k$) for which $\sum_{i \in I} e(s_i) = K$. This is an instance of the EQUAL CARDINALITY PARTITION (ECP) problem, which is known to be NP-hard [9, *version of* SP12].

Secondly, if we want to have stages with equal sizes, we can add the "blue-and-dashed" parts of Fig. 2, namely, tables F and L with $e(F) = e(L) = K$, $(F, L) \in A(T)$, and we modify stages 1 and 4 such that they have a size of $K + k$ (just like stages 2 and 3). One can see that, in this setting, T can be embedded exactly if tables f_i, s_i, l_i ($i \in \{1, \dots, 2k\}$) are embedded as in the previous case, while tables F and L are associated to stages 1 and 4, respectively. Again, this is solvable exactly if the the ECP problem has a solution with numbers $e(s_i)$ ($i \in \{1, \dots, 2k\}$).

Lastly, if we want to make the TDG a rooted connected DAG, we can add the "red-and-dashdotted" parts of Fig. 2, namely, stage 0 with size $K + k$, table R with $e(R) \leq K + k$ and arcs (r, f_i), $i \in \{1, \dots, 2k\}$ and (r, F). For a valid embedding, R should be embedded in stage 0, while the rest of the tables have to be embedded the same as before, thus this version is also equivalent to the ECP problem. Thus, 1D1R is NPC.

If *hsplit* is permitted, by adding $(F, s_i), (s_i, L)$, $i \in \{1, \dots 2k\}$ (i.e., the green-and-dotted dependencies) to $A(T)$, one can check that the *hsplit* cannot be used in a valid embedding, since it would cause not to fully utilize the capacities of a stage from among stages $1, \dots, 4$, and by doing so, by lack of stage size, some fragments of some tables could not be embedded in any stage. Thus, 1D1R-*hsplit* is in NPC.

One can observe that 1D1R-*hsplit* can be polynomially reduced to 2D1R by applying the followings: 1) taking a finite stage width w, 2) replacing each table entry with a w wide string; thus, 2D1R is in NPC. Speaking of 2D1R, it is clearly in NP. If all the tables have to be matched to the same type of resource, the problems become equivalent to 2D1R. Thus, 2D2R is also NPC. □

B INAPPROXIMABILITY OF PEP VERSIONS 1D1R, 1D1R-*HSPLIT*, 2D1R, 2D2R

The main idea behind our inapproximability results is the following. We take a pipeline embedding problem instance that can be embedded in a given number of k stages exactly if we can solve an NP-hard problem (encoded in our instance); otherwise the embedding inevitably uses at least $(k + 1)$ stages. This means that the optimal stage number cannot be approximated better than a ratio of $(k+1)/k$. We note that any approximate solution (possibly given by a PTAS) of the ecoded NP-hard problem is indifferent to this inapproximability ratio, since only an *exact* solution enables the embedding to use only k stages. For PEP version 1D1R, this $k = 2$, while for 1D1R-*hsplit* and 2D1R, $k = 4$.

Lemma 2. *PEP version 1D1R cannot be approximated better than a ratio of* $3/2$*, unless P=NP. This ratio holds both for small and large optimal stage numbers.*

Proof: Let the size of the stages be denoted by K. We have M instances of the PARTITION problem, where the sum of the numbers (representing the maximum number of entries in tables) is $2K$. From every table in the k-th instance, there is an arc to every table in the $(k + 1)$-st instance; this way the instances are placed in separate stages. If an instance has a solution, it can be embedded in 2 stages, and else it needs at least 3. This means that if all the instances have a solution, $2M$ stages are needed for the TDG, but since the PARTITION is NP-complete [9], one cannot give a solution that uses less than $3M$ stages, unless P=NP. To make the TDG contain an arborescence, one can add a table in front of the first instance, and by $M \to \infty$ the proof follows. □

Lemma 3. *PEP versions 1D1R-hsplit, 2D1R, and 2D2R cannot be approximated better than a ratio of* $5/4$*, unless P=NP. This ratio holds both for small and large optimal stage numbers.*

Proof: The proof is analogue to the proof of Lemma 2, but here we take M instances of TDG-s depicted in Subfig. 2b but without table R. We separate the stages of the instances as follows: for every $k \in 1, \dots, M - 1$ there is an arc from the l_i's and L of instance k to the f_i's and F of instance $(k + 1)$. If an instance is solvable, it takes 4 stages, otherwise, it takes 5, and it is NP-complete to distinguish between these two cases (see proof of Thm. 1). This means that if all the instances have a solution, $4M$ stages are needed for the TDG, but since the ECP is NP-complete [9], one cannot give a solution that uses less than $5M$ stages, unless P=NP. To make the TDG contain an arborescence, one can add a table in front of the first instance, and by $M \to \infty$ the proof holds for 1D1R-*hsplit*.

It is easy to see that models 2D1R and 2D2R contain model 1D1R-*hsplit* as special cases. The proof follows. □

C APPROXIMABILITY BOUNDS

The basic idea behind our algorithms and proofs of constant approximations is the following. The TDG nodes are grouped in sets $\mathcal{L}_1, \mathcal{L}_2, \dots$, such that every TDG arc is 'forward', i.e., for a TDG arc (a, b), $a \in \mathcal{L}_i$ and $b \in \mathcal{L}_j$ means $i < j$. This way, nodes of each set $\mathcal{L}_i$ can be mapped to the same stages, dealing with most of the precedence constraint issues. For all PEP versions, we prove that, among the stages used in our embedding, with at most $c_1 * \text{OPT}$ exceptions, each stage is full at least in a ratio of $1/c_2$ (where OPT is the optimal stage number). This induces a $(c_1 + c_2)$-constant approximation.

Note that we only calculate a plan of the embedding. That is, 1) in case of PEP versions 1D1R, 1D1R-*hsplit*, and 2D1R, for each table, we only store its starting stage and position, and its end stage and position, resp; and 2) in case of versions 2D2R, and 2D2R-T/S we store these values for each table for both kind of resources. Clearly, the actual embedding phase needs some additional time. Since our outputs are constant approximations of the OPT optimal stage number, this additional time scales linearly not only with the stage sizes, but with OPT too.

Claim 7. *The optimum of 1D1R can be* 3*-approximated in quasi-linear time:* $O\big(n(T)\log n(T) + m(T)\big)$.

Proof: We add a hypothetical MAT L to the TDG with arcs (L, l_i) to all source nodes of the TDG. For each j, we call the set of MATs $l_k \in V(T)$ having a longest directed path of length j from L as a level $\mathcal{L}_j$. We embed levels $\mathcal{L}_1, \mathcal{L}_2, \dots$ each after as follows. After all levels preceding $\mathcal{L}_j$ were embedded, we start a new stage s_i for the MATs in $\mathcal{L}_j$. We will embed the MATs of $\mathcal{L}_j$ in decreasing order of their size. We embed as many MATs in s_i as possible (obviously, if there is a MAT that is bigger than the stage capacities, then there is no feasible solution), and if needed, we open up new stages for the remaining MATs of the level. We claim that this gives a valid embedding. The output uses at most 3∗OPT stages since 1) not counting the last stages of the levels, each stage is at least half full (this means ≤ 2∗OPT stages), and 2) the last stages of the levels are at most ≤OPT.

Speaking of the complexity, the levels can be computed in $O\big(n(T)+m(T)\big)$, the orderings can be done in $O\big(n(T)\log n(T)\big)$ total time, then the computation of the embedding takes $O\big(n(T)+m(T)\big)$, completing the proof. □

Claim 8. *The optimum of 2D1R can be* 3*-approximated in quasi-linear time:* $O\big(n(T)\log n(T) + m(T)\big)$.

Proof: The proof of *3-approximation* of the 2D1R FFL described in the paper goes as follows. Let OPT denote the minimal number of stages the TDG can be mapped to. Let h denote the number of levels in the TDG. For each level $\mathcal{L}(i)$, let S_i and E_i denote the first and last stage where nodes of the level are matched. We claim that at least half of the memory of each row of each stage $S_i, \dots, E_i - 1$ is used. This implies $\sum_{i\in\{1,\dots,h\}} E_i - 1 - S_i \le 2 * \mathrm{OPT}$. Also, $\mathrm{OPT} \ge h$, since we need at least as many stages as the number of levels we have. These induce $E_i \le 3 * \mathrm{OPT}$. □

To decipher the *complexity* of the 2D1R FFL, we only analyse the additional calculations which have to be done compared to the 1D1R-*hsplit* FFL (that runs in $O\big(n(T)+m(T)\big)$ by Theorem 1). The ordering of tables in each level by their width can be done in $O(n(T)\log n(T))$ total time. For each table, we need constant time to compute its start and end coordinates, meaning $O(n(T))$ complexity. The proof follows. □

Claim 9. *The optimum of 2D2R can be* 8*-approximated in quasi-linear time:* $O\big(n(T)\log n(T) + m(T)\big)$. *If* $w_T \ge w_S$*, it can be* 5*-approximated in the same complexity.*

Proof: In the following, we prove that the 2D2R version of FFL is an 8-approximation in general, then we will discuss why the factor of the approximation decreases by 3 if $w_T \ge w_S$.

First, let us concentrate on a given level $\mathcal{L}_i$. Let OPT_i denote the optimal number of stages that $\mathcal{L}_i$ can be mapped to. Similarly, let ALG_i denote the number of stages that the 2D2R FFL needs for this. Let us recall that the last stage occupied by stage $\mathcal{L}_i$ is denoted by $S[E(i-1)]$. When embedding to TCAMs (also with the 2D1R version of FFL), we scale the width unit to match the width of the TCAM.

We state the followings. 1) those MATs that can be mapped only to TCAMs, for some t_i, occupy stages $S[E(i-1)+1], \dots, S[E(i-1)+t_i]$, that (apart from the last of them) have their TCAMs at least half full. 2) if the width of SRAMs is greater that the width of TCAMs (i.e. $w_S > w_T$), those MATs that can be mapped only to SRAMs due to their width, for some s_i, occupy stages $S[E(i-1)+1], \dots, S[E(i-1)+s_i]$, that (apart from the last of them) have their SRAMs at least half full. If no such MATs exist or $w_T \ge w_S$, let $s_i = 1$. 3) the remaining MATs are embedded to the TCAMs of stages $S[E(i-1)+t_i], S[E(i-1)+t_i+1], \dots$ and SRAMs of stages $S[E(i-1)+s_i], S[E(i-1)+s_i+1], \dots$, stage-by stage in an increasing stage number, and including the remaining free spaces in TCAMs of stage $S[E(i-1)+t_i]$ and SRAMs of stage $S[E(i-1)+s_i]$. We denote the last stage of this embedding of the level $S[E(i-1)+1+\max\{s_i,t_i\}+l_i]$.

When filling up the TCAM of stage $S[E(i-1)+t_i]$ with MATs that can be mapped to both memories in step 3, the last row row_{TCAM} used by step 1 may remain not half full (the same applies for SRAMs last row in step 2). Together with the last stage of the level, there can be three stages that are not completely half full. By replacing some rows, we can eliminate one of them: if there is a half-full row in the TCAM part of the last stage $S[E(i-1)+l_i]$, we replace it with row_{TCAM}. If there is no half-full row, then the last stage actually contains only one short row, which can be added to row_{TCAM} (we assume that TCAMs are filled up first in a new stage in step 3). Note that these apply indifferently to the widths of the SRAMs and TCAMs.

We can see by Claim 8 that $t = \sum_{\text{levels}}(t_i - 1) \le 2 * \mathrm{OPT}$ (since at least t TCAMs are at least half full, and the optimum embedding has to have enough TCAM capacity to store the TCAM-only tables), and similarly, $\sum_{\text{levels}}(s_i - 1) \le 2 * \mathrm{OPT}$. This means $\sum_{\text{levels}}(\max\{t_i, s_i\} - 1) \le 4 * \mathrm{OPT}$.

For similar reasons, tables that can be mapped to both TCAMs and SRAMs, fill at most another $\le 2 * \mathrm{OPT}$ stages at least half full (i.e. $\sum_{\text{levels}} l_i \le 2 * \mathrm{OPT}$), meaning at most $6 * \mathrm{OPT}$ stages in total up to this point. For each level, we have not counted i) the last stage it uses, and (maybe) ii) the last stage storing SRAM-only tables. Since there are ≤ OPT levels, we can conclude that the algorithm uses at most $(6+2) * \mathrm{OPT} = 8 * \mathrm{OPT}$ stages.

We can observe that in case of $w_T \ge w_S$, $s_i \equiv 0$ for all i, and our upper bound shrinks to $5 * \mathrm{OPT}$, because then (implicitly) SRAM-only tables do not exist, and thus they do not have to be taken in count. □

Claim 10. *The optimum of 2D2R-T/S can be* 9*-approximated in quasi-linear time:* $O\big(n(T)\log n(T) + m(T)\big)$. *If* $w_T \ge w_S$*, it can be* 6*-approximated in the same complexity.*

Proof: Let the maximal number of tables per stage be denoted by c. Let $\mathrm{OPT}_{\text{2D2R-T/S}}$ denote the optimal stage number of 2D2R-T/S for a given c, and $\mathrm{OPT}_{\text{2D2R}}$ denote the optimal stage number for $c = \infty$, which is equal to the optimum of 2D2R by the problem definition. We claim $\mathrm{OPT}_{\text{2D2R}} \le \mathrm{OPT}_{\text{2D2R-T/S}}$.

The output of 2D2R FFL can be transformed to a 2D2R-T/S-compatible embedding by dividing each stage hosting a number of $k > c$ tables to $\lceil k/c \rceil$ stages, each hosting $\le c$ MATs. Since $\mathrm{OPT}_{\text{2D2R-T/S}}$ is greater or equal to the number of stages hosting c MATs, by Claim 9, this results into $\le 8 * \mathrm{OPT}_{\text{2D2R}} + \mathrm{OPT}_{\text{2D2R-T/S}} \le 9 * \mathrm{OPT}_{\text{2D2R-T/S}}$. Furthermore, in case of $w_T \ge w_S$, also by Claim 9, the approximation bound modifies to $\le 5 * \mathrm{OPT}_{\text{2D2R}} + \mathrm{OPT}_{\text{2D2R-T/S}} \le 6 * \mathrm{OPT}_{\text{2D2R-T/S}}$, completing the proof. □