Compiling Packet Programs to dRMT Switches: Theory and Algorithms

Balázs Vass
Budapest University of Technology and Economics and ELKH-BME Information Systems Research Group

Ádám Fraknói
ELTE Eötvös Loránd University

Erika Bérczi-Kovács
Alfréd Rényi Institute of Mathematics, ELTE Eötvös Loránd University, MTA-ELTE Egerváry Research Group on Combinatorial Optimization

Gábor Rétvári
Budapest University of Technology and Economics and Ericsson Research

ABSTRACT

A critical step in P4 compilation is finding an 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 in the dRMT (disaggregated Match-Action Table) switch architecture. We report mixed results. We find that optimizing P4 program embedding for maximizing throughput is computationally intractable even when some architectural constraints are relaxed, and there is no hope for a tractable approximation with arbitrary precision unless $\mathcal{P}$=$\mathcal{NP}$. At the same time, we find that the maximal throughput embedding is approximable in quasi-linear time with a small constant bound. Our evaluations show that the proposed algorithm outperforms the heuristics of prior work both in terms of throughput and compilation speed.

CCS CONCEPTS

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

KEYWORDS

reconfigurable switches, packet programs, pipeline embedding, P4 compilation, algorithmic complexity, approximation algorithms

ACM Reference Format:
Balázs Vass, Ádám Fraknói, Erika Bérczi-Kovács, and Gábor Rétvári. 2022. Compiling Packet Programs to dRMT Switches: Theory and Algorithms. In *P4 Workshop in Europe (EuroP4 '22), December 9, 2022, Roma, Italy.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3565475.3569080

EuroP4 '22, December 9, 2022, Roma, Italy

ACM ISBN 978-1-4503-9935-7/22/12…$15.00
https://doi.org/10.1145/3565475.3569080

1 INTRODUCTION

Computing applications critically depend on more efficient, reliable, flexible, and observable networks [11]. Accordingly, programming reconfigurable switch pipelines using a high-level domain-specific language like P4 [2] is increasingly being adopted in diverse application areas, like large-scale disaggregation, in-network computation [4], telemetry [10], load-balancing [9, 12], etc. We witness dataplane programs growing in complexity, including more and larger match-action tables, diverse header parse graphs, table–action dependency relationships, and match/action types [1]. At the same time, new generations of programmable switch ASICs [7] feature 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 units, and the supported level of concurrency at each stage, find the best encoding of the P4 program into the target switch pipeline so that control and data dependencies in the program are reproduced in a semantically correct way. Here, *the "best" encoding may be such that it maximizes the throughput, while keeping the latency within reasonable bounds.*

Programmable packet forwarding ASICs: RMT vs. dRMT. The seminal paper [8] set the stage for P4 program compilation for the *Reconfigurable Match-Action Table* (RMT) switch architecture, using an abstract model to describe the resource requirements and data-/control-dependencies of P4 programs as well as the switch dataplane resources. The RMT architecture, however, has two important restrictions: (1) a table memory is local to an RMT pipeline stage, implying that memory not used by one stage cannot be reclaimed by another, and (2) RMT is hardwired to sequentially execute matches followed by actions as packets traverse pipeline stages. Thus, P4 embeddings for RMT may exhibit performance cliffs; e.g., adding only a single MAT to a P4 program may halve the throughput due to recirculation. Moreover, P4 embedding for the RMT architecture poses inherent algorithmic difficulties [16].

The *disaggregated RMT (dRMT)* architecture is a recent upgrade to RMT to address these issues [3]. First, dRMT moves table memories out of pipeline stages and into a centralized pool that is accessible through a crossbar. Second, dRMT replaces RMT's pipeline

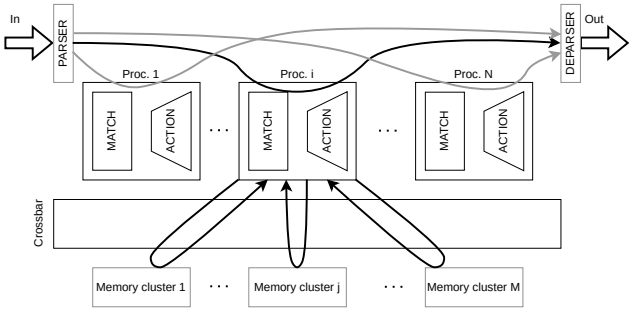
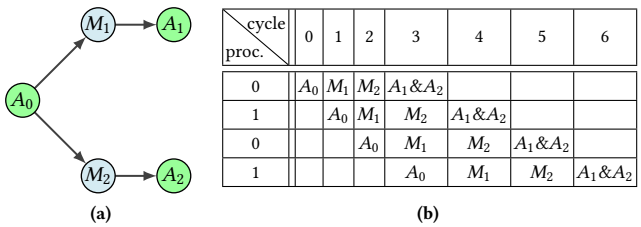


Figure 1: disaggregated RMT architecture



cycle / proc.	0	1	2	3	4	5	6
0	A_0	M_1	M_2	$A_1\&A_2$			
1		A_0	M_1	M_2	$A_1\&A_2$		
0			A_0	M_1	M_2	$A_1\&A_2$	
1				A_0	M_1	M_2	$A_1\&A_2$

Figure 2: The graph representation of a toy program (a), where A_i and M_i stand for action and match nodes/operations. Supposing a processor can initiate ≤ 1 match per clock cycle, (b) encodes an optimal embedding of the program, where $P = 2$.

stages with a cluster of processors that can execute match and action operations in any order (see Fig. 1). Intuitively speaking, and considering a single processor for a moment, the aim of maximizing throughput translates to minimizing the (average) number of clock cycles P between the termination of processing two consecutive packets. Supposing that a processor can ensure finalizing a packet in every P-th clock cycle, it is easy to see that the number of processors needed for achieving line rate (i.e., to finish the processing of a packet in each clock cycle) is also P, since processors can process packets in a round robin fashion (see Fig. 2, credits to [3]). When there are more than one processors availalbe, the *throughput is inversely proportional to the number of processors.*

At the moment, there are many open algorithmic questions related to P4 program embedding over the dRMT architecture. This is becoming increasingly troubling, since P4 compilation times can easily grow beyond practical. For instance, evaluating one of the exemplary P4 programs often cited to demonstrate the complexities of real-life P4 programs, a subprogram of `switch.p4` called "Egress", [3] identifies $P = 7$ as a firm lower bound on the complexity of the best possible P4 embedding. At the same time, it was not possible to reduce the complexity of the embedding below $P = 11$ using the heuristics and ILPs proposed in [3], leaving open the question of whether a further 57% throughput increase is possible.

Disagregated P4 pipeline embedding. Motivated by the above, in this paper *we initiate the study of the algorithmic landscape of the disaggregated P4 pipeline embedding problem (DPEP)*, where the aim is to find a valid P4 program embedding that maximizes the throughput, or equivalently, minimizes P. To the best of our knowledge, ours is the first principled approach to this end. We take off from a simplified model of a programmable switch target (the BASIC model) on which pipeline embedding maps to a well-known combinatorial optimization problem that can be solved to optimality in polynomial time. Then, *we re-introduce additional degrees of complexity* into the model to obtain increasingly more realistic and restrictive models, eventually reproducing the problem formulations of [3] in the models called WIDTH-IPC1 and WIDTH-IPC2. Thus, *we give a comprehensive characterization of the respective computational complexity and approximation bounds* and cast several open problems.

The main contributions of the paper are the following.

(1) We fix the algorithmic complexity of different versions of the DPEP problem, and we present low-polynomial constant-approximation algorithms. These results are summarized in Table 1.
(2) Our evaluations show that one of our P4 embedding algorithms, Alg. 1, achieves at least 85% of the theoretically optimal throughput on all P4 programs studied in [3], significantly improving on the heuristic `rnd_sieve` of [3] that achieves only 73% of the theoretical optimum.
(3) We provide upper bounds on the achievable P, a feature that has not been proposed in previous approaches.

The rest of the paper is organized as follows. Sec. 2 introduces the simplified DPEP problem variants, alongside stating the algorithmic complexities and some related inapproximability and approximability bounds. Then, Sec. 3 presents the preliminary simulation results of our algorithms. Finally, the conclusions are drawn in Sec. 4. Formal proofs of our statements are provided in the Appendix.

2 ALGORITHMIC ISSUES OF P4 PROGRAM EMBEDDING TO DRMT SWITCHES

In this section, we build a sequence of increasingly complex models to characterize the resource requirement for embedding P4 programs into the dRMT pipeline. 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 all of our models, the P4 program is modeled by an Operation Dependency Graph (ODG, [3]) $D = (V, E)$, $V = V_A \cup V_M$, where disjoint set of vertices V_A and V_M represent the match and action nodes, respectively, and arc set E encodes the inter-dependency between the vertices. If the tail of an arc $e = (u, v)$ is a match or action node, then the execution of v can start at least ΔM or ΔA cycles after the start of execution of u, respectively. Moreover, in each CPU cycle, each processor can initiate up to $\overline{M}$ parallel table searches, and can modify up to $\overline{A}$ action fields in parallel. Parameters ΔM, ΔA, $\overline{M}$ and $\overline{A}$ are positive integers. For example, the setting on Fig. 2 can be described by $\Delta M = \Delta A = \overline{M} = 1$, and an arbitrary $\overline{A} \geq 2$.

Also, in line with [3], we restrict our study to cyclic dRMT schedules, where a single packet processing plan is repeated on all packets processed by all processors (cf. [3, Sec. 3.2.]). To give an intuition behind or positive (approximability) results, we anticipate that, based on [3, Theorem 3.5], the dRMT scheduling problem can be simplified to the problem of scheduling a single packet on a single

Model name:	BASIC	IPC1	WIDTH	WIDTH-IPC1	WIDTH-IPC2
New feature on top of the basic constraints	(basic model)	Max. 1 packet per processor per cycle (IPC= 1)	arbitrary table widths	arbitrary table widths + IPC= 1	arbitrary table widths + IPC= 2 ($\leq$2 pkt./proc./cycle)
Complexity	$\mathcal{P}$	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard	?
Bad news: Inapproximable better than ... (unless $\mathcal{P}$=$\mathcal{NP}$)	OPT	4/3∗OPT	3/2∗OPT	3/2∗OPT	?
Good news: Constant approximable in...	OPT	3∗OPT	?	4∗OPT	8∗OPT

Table 1: Overview of the main results. Bad news: the Disaggregated Pipeline Embedding Problem (PEP) is $\mathcal{NP}$-hard even with relaxing some constraints. Good news: the DPEP is polynomially solvable under the BASIC model, and is constant approximable in quasi-linear time even when considering the model tackled by [3].

processor. This single packet scheduling has to fulfill a requirement of *P-periodicity*: the set of nodes assigned to clock cycles t, $t+P$, $t+2P$, ... must meet the ΔM, ΔA, $\overline{M}$, $\overline{A}$ (and later on the width and inter-packet concurrency) requirements together, for all $t \in \{1,\ldots,P\}$.

2.1 BASIC: A simplified model

In the BASIC model, there are no additional constraints to those described above. Every table has a unit width. It is clear that the minimal value of P is at least the maximum of $\lceil |V_m|/\overline{M} \rceil$ and $\lceil |V_a|/\overline{A} \rceil$. As it turns out, the maximum of these two values is reachable with a simple greedy algorithm (see Theorem 1).

Results for this model. *DPEP under the BASIC model can be solved to optimality in polynomial time.*

2.2 IPC1: Inter-packet concurrency

On top of the constraints of BASIC, in the IPC1 model, we assume that each processor may start a match for at most a fixed number (*Inter-Packet Concurrency*, IPC) of different packets and likewise start actions for up to IPC different packets. The set of packets that start matches and the set of packets that start actions need not be equal. Below we assume IPC=1. It turns out that in the presence of the IPC constraint, the problem becomes not only $\mathcal{NP}$-hard, but there is also no hope for a polynomial time approximation scheme (PTAS) for P (unless $\mathcal{P}$=$\mathcal{NP}$). The $\mathcal{NP}$-hardness and inapproximability can be reduced to a well-known scheduling problem (see Theorem 2). On the bright side, in this setting, there exists a 3-approximation algorithm, described in Theorem 6.

Results for this model. *DPEP under the IPC1 model is $\mathcal{NP}$-hard. Bad news: the optimal number of cores to achieve line rate cannot be approximated better than* 4/3 *unless $\mathcal{P}$=$\mathcal{NP}$. Good news: the optimum can be* 3*-approximated in linear time:* $O(|V|+|E|)$.

2.3 WIDTH: Variable table widths

Next, on top of BASIC we will also allow each match and action node to be of arbitrary width, measured by a positive integer $W : V \to \mathbb{N}^+$. We represent this in our WIDTH model by letting each processor to initiate up to $\overline{M}$ parallel unit-wide (say, b bits) table searches in each cycle; e.g. a look up on two match-action tables with key sizes 3 and 2, respectively, equals five parallel lookup vectors of b bits each, as long as $5 \leq \overline{M}$. It turns out that introducing variable table (key) widths on top of the BASIC model also makes the DPEP $\mathcal{NP}$-hard and inapproximable (see Theorem 3.):

Results for this model. *DPEP under the WIDTH model is $\mathcal{NP}$-hard. Bad news: the optimal number of cores to achieve line rate cannot be approximated better than* 3/2 *unless $\mathcal{P}$=$\mathcal{NP}$.*

2.4 WIDTH-IPC1: Full-blown dRMT model

Our next model, WIDTH-IPC1, is equivalent to the one studied in [3]. Here, we simultaneously require IPC= 1 and allow arbitrary table widths. As expected, combining additional constraints does not make the problem easier: the minimal P for which an embedding exists cannot be approximated better than 3/2 (unless $\mathcal{P}$=$\mathcal{NP}$, see Theorem 2). As a promising positive result, though, we show that in WIDTH-IPC1 the optimum can be 4-approximated in quasi-linear time; see Alg. 1 (see Theorem 5). The algorithm is based on the observation that the optimal period for a scheduling solution (see Definition 1) is independent of values ΔM and ΔA, because it depends only on the *number* of clock cycles with at least one match/action node (see Lemma 1). Our algorithm greedily finds a solution with $\Delta M = \Delta A = 1$ (a *pre-scheduling*, see Definition 2) such that clock cycles are filled with match/action nodes at least half full when possible, resulting a 4-approximation.

Results for this model. *DPEP under the WIDTH-IPC1 model is $\mathcal{NP}$-hard. Bad news: the optimal number of cores to achieve line rate cannot be approximated better than* 3/2 *unless $\mathcal{P}$=$\mathcal{NP}$. Good news: the optimum can be* 4*-approximated in quasi-linear time:* $O(|V| \log |V| + |E|)$.

2.5 WIDTH-IPC2: Loose IPC constraints

The original paper [3] also considers the case when IPC is 2, possibly allowing more compact program embeddings. Intuitively speaking, increasing IPC from 1 to 2 may allow at most twice as efficient embeddings. Thus, the the greedy algorithm of model WIDTH-IPC1 will give an 8-approximation in the WIDTH-IPC2 model (see Theorem 7).

Results for this model. *Good news: the optimum can be* 8*-approximated in quasi-linear time:* $O(|V| \log |V| + E)$.

For IPC= 2 the ILP solvers of [3] can compute efficient program embeddings relatively easily. Thus, we will not study this model further here.

Algorithm \ Graph	Egress $\|V\| = 104$ $\|E\| = 291$	Ingress $\|V\| = 224$ $\|E\| = 930$	Combined $\|V\| = 328$ $\|E\| = 1221$
rnd_sieve i.e., [3]-greedy	13	21	30
Our greedy	13	19	23
[3] ILP	11	17	21
ILP lower bound	7	15	21

Table 2: Best P values computed by different algorithms

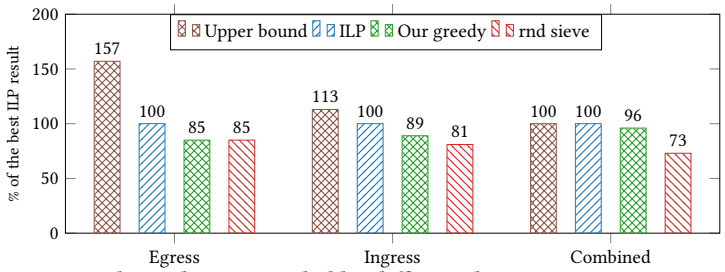


Figure 3: Throughput provided by different heuristics as precentage of the best ILP solution

Algorithm 1: WIDTH-IPC1 Our Greedy

Input: ODG $D = (V, E)$; $W : V \to \mathbb{N}^+$; $\overline{M}, \overline{A}$
Output: $PS : V \to \mathbb{N}^+$
begin
1 $i := 1$; $V' := V$
2 **while** $V' \neq \emptyset$ **do**
3 a := list of action nodes with 0 indegrees, descending order of width
4 m :=list of match nodes with 0 indegrees, descending order of width
5 w_a := sum of widths in a
6 w_m := sum of widths in m
7 current_usage := 0
8 **if** $w_m \geq 1/2\overline{M}$ *and* $w_a \geq 1/2\overline{A}$ **then**
9 Go to line 12 or 19
10 **if** $w_a \geq 1/2\overline{A}$ *and* $w_m < 1/2\overline{M}$ **then**
11 Go to line 19
12 **while** $m[0]$+*current_usage* $\leq \overline{M}$ **do**
13 current_usage += $m[0]$
14 PS$[m[0]] := i$
15 $V' := V' \setminus \{m[0]\}$
16 $m := m - m[0]$
17 i := i+1
18 **if** $w_m \geq 1/2\overline{M}$ **then**
continue
19 **while** *a[0]+current_usage* $\leq \overline{A}$ **do**
20 current_usage += $a[0]$
21 PS$[a[0]] := i$
22 $V' := V' \setminus \{a[0]\}$
23 $a := a - a[0]$
24 $i := i + 1$
25 **return** PS

3 PRELIMINARY SIMULATION RESULTS

In this section, we present our simulation studies on P4 embeddings for the dRMT architecture over the WIDTH-IPC1 model. Our goal is to maximize throughput while keeping latency under control. Running times were measured on a commodity laptop, with 64 GB RAM and 24 threads, at 2.40GHz. The code used in the evaluation is available on GitHub (https://github.com/fraknoiadam/drmt).

Maximizing the throughput. The throughput of a dRMT switch is inversely proportional to the number of processors P needed to achieve line rate [3]. Table 2 summarizes the lowest P values computed by different algorithms. In summary, Alg. 1 uses at most 19% more processors than the best ILP solution, compared to the at most > 36% extra processors used by the heuristic rnd_sieve of [3]. Recall, Alg. 1 is a *provably* constant approximation on the optimal P. In addition, the running time of Alg. 1 on the "Egress", "Ingress", and "Combined" instances obtained from `switch.p4` [3] was 7 ms, 24 ms, and 41 ms, respectively, which is beyond an order of magnitude improvement over rnd_sieve [3]. The average running time of Alg. 1 and rnd_sieve on these graphs were 0.007, 0.28, 10.5 and 0.3, 1.5, 2.7 `[sec]`, respectively. Out of 1000 runs, Alg. 1 reached the theoretically optimal P values 1000, 85, and 4 times, exploiting the fact that, in Alg. 1 there are multiple steps where random choices are made (e.g., at lines 3 and 4).

Fig. 3 visualizes the throughput provided by our greedy algorithm, and rnd_sieve [3] as the percentage of the best throughput provided by the optimal ILPs. For the Egress, Ingress, and Combined instances, Alg. 1 achieves 85%, 89%, and 96%, while rnd_sieve yields 85%, 81%, and 73%, respectively. In other words, our algorithm performs at least as well as the rnd_sieve. Moreover, in these cases, with the size of the input graph growing, Alg. 1 got closer to the best throughput computed by the ILP formulation, while the relative performance of rnd_sieve degraded.

Running times and latency. For each of the three program instances, Table 3 shows the optimal latency ($T = T_P$) in the case of the three lowest P values that could be computed by the ILPs of the paper. We can see that, while approaching the best P obtained by the ILP, the optimal latency remained more or less steady in the case of Ingress and Combined, and, for Egress, it grew only by less than 5% also. Running times for achieving the optimal T values did not grow radically either on the example cases, except for Egress. We can conclude that, contrary to the intuition, a higher throughput (i.e., a lower P) has no significant impact on the lowest latency (T) achievable.

4 CONCLUSION AND FUTURE WORK

P4 pipeline embedding sits at the core of programmable dataplane use cases, as a crucial step for deploying P4 programs to real targets. Given the booming data rates and sizes of P4 programs, maximizing throughput is of paramount importance. In this paper, we analyze the theoretical aspects of this problem, characterizing its complexity and providing bounds for different models. Our results may help

	Egress			Ingress			Combined		
P	11	12	13	17	18	19	21	22	23
optimal T_P	217	208	206	245	246	244	243	244	243
time `[sec]`	1203	76	107	106	59	23	118	25	109

Table 3: P vs T: higher throughput does not mean higher latency.

judge the performance of existing P4 compilers and build better compilers for future switch architectures.

In our future work, we intend to analyze the adaptability of different existing P4 program deployment approaches [5, 6, 14, 15] for taking advantage of the dRMT's architectural enablers in throughput maximization. Furthermore, we aim to tighten our lower and upper bounds for the different DPEP variants and provide a clear picture of how existing greedy approaches compare to an optimal cyclic dRMT scheduling. Another interesting direction is unfolding the possible and practical benefits and drawbacks of enabling a number of $l > 1$ packet scheduling cycles in dRMT scheduling.

A FORMAL PROOFS

A.1 Formal problem statement

Suppose that one of the DPEP model inputs is given with input parameters $D = (V, E)$, $\Delta M, \Delta A, \overline{M}, \overline{A}, IPC \in \{1, 2, \infty\}$ and $W : V \to \mathbb{N}$. For brevity we will use the latency function $l : V \to \{\Delta M, \Delta A\}$, where $l(v) = \Delta M$ or ΔA if v is a match/action node, respectively.

Definition 1. *A **scheduling** of the nodes is a function $S : V \to \mathbb{N}^+$ such that for every arc $(v_i, v_j) \in E$ we have $S(v_j) - S(v_i) \geq l(v_i)$.*

*For a scheduling S and period $P \in \mathbb{N}^+$, let $\mathcal{S}_P$ denote the set of schedulings S_i such that $S_i(v) = S(v) + iP$ (for $i \in \mathbb{N}$). We say that a scheduling S is **feasible with period** P if*

(1) $\forall t \in \mathbb{N}^+ : \sum_{S_i \in \mathcal{S}_P} \sum_{\substack{v_m \in V_m \\ S_i(v_m)=t}} W(v_m) \leq \overline{M}$

(2) $\forall t \in \mathbb{N}^+ : \sum_{S_i \in \mathcal{S}_P} \sum_{\substack{v_a \in V_a \\ S_i(v_a)=t}} W(v_a) \leq \overline{A}$

(3) $\forall t \in \mathbb{N}^+ : \#\left\{S_i \in \mathcal{S}_P \middle| \exists v_m \in V_m : S_i(v_m) = t\right\} \leq IPC$

(4) $\forall t \in \mathbb{N}^+ : \#\left\{S_i \in \mathcal{S}_P \middle| \exists v_a \in V_a : S_i(v_a) = t\right\} \leq IPC.$

*In a **DPEP** instance, the goal is to find the minimum P such that there exists a scheduling S which is feasible with period P. The decision version of DPEP is to decide for a given value k if there exists a feasible P-periodic scheduling with $P \leq k$.*

A.2 Complexity

Theorem 1. *For model BASIC $P = \max\left(\left\lceil \frac{|V_m|}{\overline{M}} \right\rceil, \left\lceil \frac{|V_a|}{\overline{A}} \right\rceil\right)$ is the optimal period, and a feasible P-periodic scheduling can be found in polynomial time, in $O\big(|E| + |V|P\big)$.*

Proof. It is clear that $\left\lceil \frac{|V_m|}{\overline{M}} \right\rceil$ and $\left\lceil \frac{|V_a|}{\overline{A}} \right\rceil$ are lower bounds for P. To prove the other direction, let $v_1, v_2, \ldots, v_n$ be an arbitrary topological order of the nodes (i.e. $i < j$ if $v_i v_j \in E$). We will construct a scheduling S of the nodes in this order in the following way. $S(v_1) := 1$. For $j > 1$, let $\delta_j := \max\{S(v_i) + l(v_i) \mid v_i v_j \in E\}$ if v_j has at least one entering arc, otherwise $\delta_j := S(v_{j-1})$. If $v_j \in V_m$, let $S(v_j) := \min\{k \geq \delta_j \mid \#\{i : i < j, v_i \in V_m, S(v_i) \equiv k \mod P\} \leq \overline{M}\}$. Similarly, if $v_j \in V_a$, let $S(v_j) := \min\{k \geq \delta \mid \#\{i : i < j, v_i \in V_a, S(v_i) \equiv k \mod P\} \leq \overline{A}\}$. Note that by choice of P, the set to be minimized is never empty. The total number of steps for calculating all δ_i values is $O(|E|)$, whereas to determine a minimum value for an $S(v_j)$ we need to check at most P values from $\delta_j, \delta_j + 1, \ldots, \delta_j + P - 1$, giving a running time of $O\big(|E| + |V|P\big)$. □

Theorem 2. *The decision versions of models IPC1 and WIDTH-IPC1 are NPC. Furthermore, they cannot be approximated better than a ratio of $4/3$ unless $\mathcal{P}$=$\mathcal{NP}$.*

Proof. The problem can be reduced to the scheduling problem $P|prec, p_j = 1|C_{max}$ that can be reduced to CLIQUE problem [13]. □

Theorem 3. *The decision version of model WIDTH is NPC, and it cannot be approximated better than a ratio of $3/2$ unless $\mathcal{P}$=$\mathcal{NP}$.*

Proof. Assume we have only match nodes and $\Delta M = 1$. In this case, the problem is equivalent to the Pipeline Embedding Problem (PEP) variant 1D1R [16]. □

Theorem 4. *The decision version of model WIDTH-IPC1 cannott be approximated better than a ratio of $3/2$, unless $\mathcal{P}$=$\mathcal{NP}$.*

Proof. It can be reduced to PEP version 1D1R [16]. □

A.3 Approximation algorithms

In this subsection, we describe approximation algorithms for models IPC1, WIDTH-IPC1, and WIDTH-IPC2. The key idea is to find a proper partial order of the nodes that can be expanded into a scheduling.

Definition 2. *A function $PS : V \to \mathbb{N}^+$ is a **pre-scheduling**, if*

(1) $PS(v_m) \neq PS(v_a)$ for every $v_m \in V_m, v_a \in V_a$,
(2) $PS(v_j) - PS(v_i) \geq 1$ for every arc $(v_i, v_j) \in E$,
(3) if $PS^{-1}(k) = \emptyset$ for a $k \in \mathbb{N}^+$, then $PS(v) < k$ for every $v \in V$.
(4) $\forall t \in \mathbb{N}^+ : \sum_{\substack{v_m \in V_m \\ PS(v_m)=t}} W(v_m) \leq \overline{M}$,
(5) $\forall t \in \mathbb{N}^+ : \sum_{\substack{v_a \in V_a \\ PS(v_a)=t}} W(v_a) \leq \overline{A}$.

*Let $L(PS)$ denote the **length** of the pre-scheduling, so the largest clock cycle that has an embedded node:*

$$L(PS) = \max\{i | PS^{-1}(i) \neq \emptyset\}.$$

Let A denote the number of clock cycles with at least one embedded action node. Formally, $A := \#\{i \in \mathbb{N}^+ | PS^{-1}(i) \cap V_a \neq \emptyset\}$. We define M similarly with match nodes.

*A scheduling S is an **expansion of a pre-scheduling** PS if there exists a strictly monotone function $f : \mathbb{N}^+ \to \mathbb{N}^+$ such that $S(v) = f(PS(v))$.*

Claim 1. *Every pre-scheduling has an expansion.*

Proof. We determine values $f(1), \ldots, f(L(PS))$ in this order. Let $f(1) = 1$. For $1 < i \leq L(PS)$, if there is no arc entering nodes in $PS^{-1}(i)$, then $f(i) := f(i-1) + 1$. Else let $f(i) := \max\{f(PS(v)) + l(v) | vw \in E, PS(w) = i\}$. □

Lemma 1. *Let PS be a pre-scheduling and let $IPC = 1$. For $P := \max(A, M)$ there exists an expansion of PS that is feasible with period P. Moreover, P is the smallest among such periods.*

Proof. It is easy to see that values A and M are lower bounds for the period of an expansion because the resulting scheduling has the same number of match/action clock cycles.

Now we show that PS has a feasible P-periodic expansion.

We have seen in Claim 1 that PS has an expansion. We use a similar approach to get a feasible P-periodic scheduling. In addition, we will make sure that there are no two clock cycles with the same type of nodes embedded into the same residue class modulo P, which will guarantee constraints (1)-(4) of a feasible scheduling.

Let $f(1) = 1$ and for $1 < i \leq L(PS)$ we do the followings. If there exists an arc entering a node in $PS^{-1}(i)$ then $\delta := \max\{f(PS(v)) + l(v) | vw \in E, PS(w) = i\}$, otherwise $\delta := f(i-1) + 1$.

$$f(i) := \min\{k \geq \delta \mid \nexists j < i : f(j) \equiv k \mod P \text{ and } PS^{-1}(i), PS^{-1}(j) \text{ have the same type } \} \quad (1)$$

Note that the set we are minimizing for $f(i)$ is not empty since $P \geq M$ and $P \geq A$, and former clock cycles of the same type cannot cover all residue classes modulo P. □

Theorem 5. *There is a 4-approximation algorithm for model WIDTH-IPC1.*

Proof. Based on Lemma 1, our goal is to find a pre-scheduling PS where we minimize $\max(A, M)$. Our algorithm uses a greedy approach and embeds at least half full clock cycles as long as it is possible (see Algorithm 1).

The algorithm maintains the subset V' of nodes that need to be embedded. At the beginning, let $V' := V$. Let m/a denote the current list of match/action nodes of zero indegree in the subgraph spanned by V', sorted in a descending order according to their width. At one phase of the algorithm, we embed some nodes from m and a to one or two clock cycles and then move to the next clock cycle and the next phase, when m and a are updated again. Let i denote the current first empty clock cycle.

Let w_m and w_a denote the sum of widths of nodes in m and a, respectively. In one phase, we do the following:

If $w_m < \overline{M}/2$ and $w_a < \overline{A}/2$, we embed all nodes in m to clock cycle i and all nodes in a to clock cycle $i+1$. We move on to the next clock cycle: $i := i + 2$.

If $w_m \geq \overline{M}/2$ and $w_a < \overline{A}/2$, we greedily embed only nodes in m in clock cycle i as long as possible, and move to the next clock cycle: $i = i + 1$.

Similarly, if $w_m < \overline{M}/2$ and $w_a \geq \overline{A}/2$, we greedily embed nodes in a in clock cycle i as long as possible, and then move on to the next phase and the next clock cycle.

Finally, if both $w_m \geq \overline{M}/2$ and $w_a \geq \overline{A}/2$, we can choose m or a arbitrarily and embed nodes again greedily as before into clock cycle i.

Now we prove 4-approximation. We partition the clock cycles into four groups: let HF_m/HF_a denote those clock cycles that are at least half full with match/action nodes, respectively, and similarly, let $NHF_m/\ NHF_a$ denote the list of those that are not half full. Note that $|NHF_m| = |NHF_a|$ and $NHF_m[j] = NHF_a[j] - 1$. We can assume that $M \geq A$. From Lemma 1, we get that the constructed pre-scheduling can be expanded into feasible scheduling with period M. Let P_o denote the optimal period for the problem. We know that $P_o \geq \sum_{v \in V_m} W(v)/\overline{M}$. Since $\sum_{v \in V_m} W(v)/\overline{M} \geq \sum_{v \in PS^{-1}(HF_m)} W(v)/\overline{M} \geq |HF_m|/2$ and we get $|HF_m| \leq 2P_o$.

Claim 2. *For every node v embedded in $NHF_m[i]$ or $NHF_a[i]$ ($i \geq 2$) there is a path from a node embedded in $NHF_m[i-1]$ or $NHF_a[i-1]$ to v.*

Proof. Let us consider the phase when $m = NHF_m[i-1]$ and $a = NHF_a[i-1]$. Observe that every node in the current V' is reachable from nodes embedded in $NHF_m[i-1]$ or $NHF_a[i-1]$. □

Claim 3. *There is a path of length of at least $|NHF_m|$ in D.*

Proof. Applying Claim 2 backwards starting from an arbitrary node $v_{|NHF_m|}$ embedded in $NHF_m[|NHF_m|]$ or $NHF_a[|NHF_m|]$ we get a path from a node $v_{|NHF_m|-1}$ embedded in $NHF_m[|NHF_m|-1]$ or $NHF_a[|NHF_m|-1]$, and so on. By concatenating these paths, we get a path $\mathcal{P}$ which is required in the claim. □

Note that for any path Q we have that $|V(Q) \cap V_m| \leq P_o$ and $|V(Q) \cap V_a| \leq P_o$ so $|V(Q)| \leq 2P_o$. Hence $|NHF_m| \leq |V(\mathcal{P})| \leq 2P_o$ and so $|M| = |HF_m| + |NHF_m| \leq 4P_o$, which proves the theorem. The running time of the algorithm is $O(|V| \log |V| + |E|)$. □

Theorem 6. *Model IPC1 can be 3−approximated in polynomial time.*

Sketch of proof. We can simplify the previous algorithm in Theorem 5 the following way: we do not need to sort the elements in lists m and a because all have unit width. Moreover, we apply limits $\overline{M}$ and $\overline{A}$ for embeddings instead of $\overline{M}/2$ and $\overline{A}/2$, and embed nodes to get full clock cycles (with either match or action nodes only). Let F_m and F_a denote the set of full clock cycles. We can derive a sharper bound $|F_m| \leq P_o$, which gives a 3-approximation. □

Finally, we can derive an approximation algorithm for the WIDTH-IPC2 model from the one given for the WIDTH-IPC1 .

Theorem 7. *Model WIDTH-IPC2 can be 8−approximated in polynomial time.*

Proof. Let P_1^{opt} and P_2^{opt} denote the optimal periods for WIDTH-IPC1 and WIDTH-IPC2, respectively for a pair of models with the same input parameters (except *IPC*). Since a feasible P-periodic one for WIDTH-IPC2 can be transformed into a feasible $2P$-periodic scheduling for WIDTH-IPC1, so $P_1^{opt} \leq 2P_2^{opt}$. Let P^* denote the period of the scheduling given by the 4−approximation algorithm for WIDTH-IPC1 (Theorem 5). Then $P^* \leq 4P_1^{opt} \leq 8P_2^{opt}$. □

ACKNOWLEDGEMENTS

This research was partially supported by the National Research, Development and Innovation Fund of Hungary (grant No. 135606 and FK 132524). Supported by the ÚNKP-22-4-II-BME-248 New National Excellence Program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund. Application Domain Specific Highly Reliable IT Solutions" project has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme TKP2020-NKA-06 (National Challenges Subprogramme) funding scheme.

REFERENCES

[1] Switch.p4. https://github.com/p4lang/switch/blob/master/p4src/switch.p4. Accessed: 2020-08-19.

[2] 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.

[3] Chole, S., Fingerhut, A., Ma, S., Sivaraman, A., Vargaftik, S., Berger, A., Mendelson, G., Alizadeh, M., Chuang, S.-T., Keslassy, I., et al. dRMT: Disaggregated Programmable Switching. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2017), pp. 1–14.

[4] Dang, H. T., Canini, M., Pedone, F., and Soulé, R. Paxos made switch-y. vol. 55, ACM SIGCOMM, pp. 19–24.

[5] 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.

[6] 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.

[7] Intel. Intel Tofino 3: Product Brief. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-3-brief.html. Accessed: 2022-09.

[8] Jose, L., Yan, L., Varghese, G., and McKeown, N. Compiling packet programs to reconfigurable switches. In *USENIX NSDI* (2015), pp. 103–115.

[9] 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.

[10] Kim, C., Sivaraman, A., Katta, N. P., Bas, A., Dixit, A., and Wobker, L. J. In-band network telemetry via programmable dataplanes.

[11] McKeown, N. Programmable forwarding planes are here to stay. In *ACM SIGCOMM NetPL* (2017).

[12] 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.

[13] Michael L., P. *Scheduling Theory, Algorithms, and Systems*. Springer, New York, 2012.

[14] Sivaraman, A., Cheung, A., Budiu, M., Kim, C., Alizadeh, M., Balakrishnan, H., Varghese, G., McKeown, N., and Licking, S. Packet transactions: High-level programming for line-rate switches. In *Proceedings of the 2016 ACM SIGCOMM Conference* (2016), 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.

[16] Vass, B., Bérczi-Kovács, E., Raiciu, C., and Rétvári, G. *Compiling Packet Programs to Reconfigurable Switches: Theory and Algorithms*. Association for Computing Machinery, New York, NY, USA, 2020.