Charting the Complexity Landscape of Compiling Packet Programs to Reconfigurable Switches

Balázs Vass, Erika Bérczi-Kovács, Ádám Fraknói, Costin Raiciu, Gábor Rétvári

***Abstract*—P4 is a widely used Domain-specific Language for Programmable Data Planes. 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 for RMT and dRMT target architectures. We report mixed results: we find that P4 compilation is computationally hard even in a severely relaxed formulation, and there is no polynomial-time approximation of arbitrary precision (unless $\mathcal{P}$=$\mathcal{NP}$), while the good news is that, despite its inherent complexity, P4 compilation is approximable in linear time with a small constant bound even for the most complex, nearly real-life models.**

I. Introduction

Future computing applications critically depend on more efficient, reliable, flexible, and observable networks [3]. 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 [5], telemetry [6], load-balancing [7], [8], 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 [9]. At the same time, new generations of programmable switch ASICs [10] feature even more dataplane resources and pipeline stages.

Balázs Vass and Gábor Rétvári are with Department of Telecommunications and Artificial Intelligence, Faculty of Electrical Engineering and Informatics (VIK), Budapest University of Technology and Economics (BME) and HUN-REN-BME Information Systems Research Group. Balázs Vass is also affiliated to Faculty of Mathematics and Computer Science, Babeș-Bolyai University, Cluj Napoca, Romania. Contact them on {balazs.vass,retvari}@tmit.bme.hu.
G.R. was partially funded by the grant NKFIH/OTKA Project #135606. Supported by the ÚNKP-23-4-II-BME-345 New National Excellence Program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund.

Erika Bérczi-Kovács and Ádám Fraknói are with ELTE Eötvös Loránd University, Budapest, Hungary. E. B.-K. is also with HUN-REN-ELTE Egerváry Research Group on Combinatorial Opt.. Contact them on erika.berczi-kovacs@ttk.elte.hu, and fraknoiadam@student.elte.hu. This research was in part supported by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund, financed under ELTE TKP 2021-NKTA-62, FK 132524 funding schemes, and by the János Bolyai Research Scholarship of the Hungarian Academy of Science.

Costin Raiciu is with University Politehnica of Bucharest and Broadcom. Contact him on costin.raiciu@cs.pub.ro. This research was partly funded by a Huawei Research Gift and by a VMWare Research Grant.

Parts of this work were presented on EuroP4 '20 and '22 [1], [2].

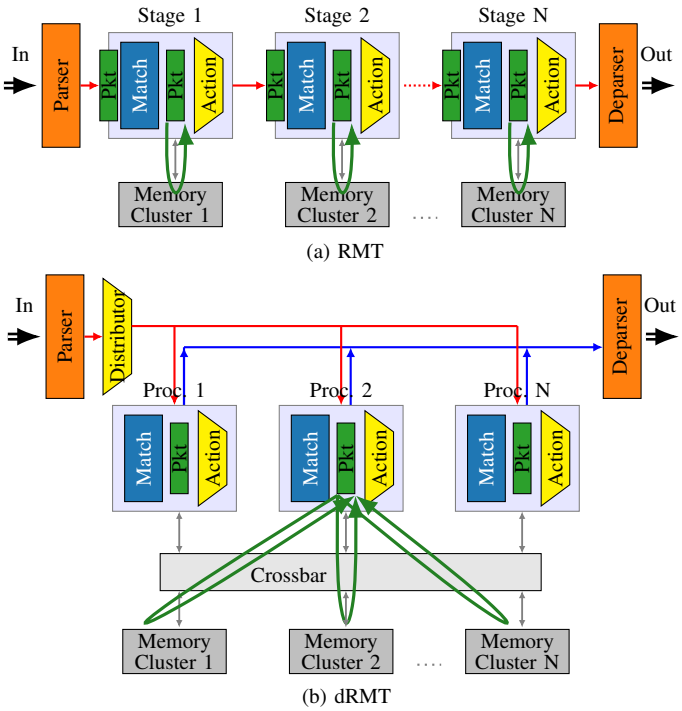


Fig. 1: Programmable packet forwarding ASICs: RMT and dRMT

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* [11]–[20]. 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.[1] 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 [17] set the stage for P4 program compilation for the *Reconfigurable Match-Action Table* (RMT) switch architecture, using an abstract model to describe the resource

[1]Higher level languages like μP4 [21] and Lyra [14] have been proposed to raise the abstraction level. They still have to solve the pipeline embedding problem before deployment, and currently use greedy approaches. Chipmunk [20], to tame complexity, 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 outcomes.

Pipeline	#nodes	#arcs	ILP runtimes in [sec] for different gaps					Greedy	Gap (greedy
			50%	30%	20%	10%	0%	runtime	#stages to opt.)
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 I: Running time of the algorithms of [17] for pipeline embedding on a commodity quad-core computer with CPU @ 2.3 GHz and 8 GB RAM. The ILP used parallel B&C on 4 threads. While the greedy approach scales with the network size, ILP runtimes explode.

requirements and data-/control-dependencies of P4 programs as well as the switch dataplane resources. By assuming the ability to recirculate packets back into the pipeline in RMT while reducing throughput by a factor of the number of recirculations, we can calculate the throughput $t_{\text{RMT}}(N)$ as a function of the number of processors N and the number of stages S required to run the program at one packet per clock cycle as (cf. [22, Eq. (9)]):

$$t_{\text{RMT}}(N) = \min\{1, 1/\lceil S/N \rceil\}. \quad (1)$$

Thus, in RMT, the task of maximizing the throughput translates to minimizing the number of stages used by the P4 program embedding.

RMT has two important restrictions: (1) a table memory is local to a 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 (see also §II-A). The *disaggregated RMT (dRMT)* architecture [22] is an upgrade to RMT addressing these issues. 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 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. We note that depending on resource constraints, this minimal period P is a natural number possibly greater than 1. 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). With a given number of N processors available, the *throughput is inversely proportional to period* P (cf. [22, Eq. (11)]). That is,

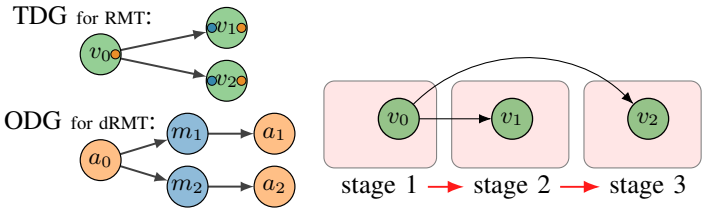
$$t_{\text{dRMT}}(N) = \min\{1, N/P\}. \quad (2)$$

This means that in dRMT, the task of maximizing the throughput translates to minimizing the period P. We note that, with similar resources, $t_{\text{dRMT}} \geq t_{\text{RMT}}$ [22, Thm. 3.3].

Algorithmic difficulties of pipeline embedding. For **RMT** architectures, [17] proposes an Integer Linear Program (ILP) to obtain an optimal solution, but with possibly exponential running time and/or memory footprint, as well as a greedy heuristics to get quick embeddings but with unknown optimality gap (margin from the optimal solution). Unfortunately, the most important algorithmic questions related to pipeline embedding have remained open since this seminal paper [17]. 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 beyond practical: Table I reports the typical pipeline embedding times on some common-case P4 programs obtained using the framework in [17], [23]. 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 are guaranteed to run in polynomial time, may produce low-quality embeddings, and they may easily conclude that there is no feasible embedding even when there is one.

P4 program embedding over the **dRMT** architecture also faces many open algorithmic questions. 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", [22] identifies as a firm lower bound that a processor cannot cope with packets arriving in less than every $P = 7^{\text{th}}$ clock cycle. At the same time, it was not possible to reduce the period P of the embedding below $P = 11$ using the heuristics and ILPs proposed in [22] in around 10^5 seconds. Since throughput is inversely proportional to period P, this leaves 57% throughput gap between the hypothetical optimal embedding and the currently known best solution. Due to the potential infeasibility of the ILP runtimes, greedy heuristics were also investigated. Here, in our experiments, our greedy embedding algorithm provided in [2] achieved at least the throughput yielded by the embedding given by the rnd_sieve of [22]. More precisely, in [2], we compared the performance of these algorithms on P4 program instances "Egress", "Ingress", and "Combined" obtained from `switch.p4` [22], measured as the percentage of the best throughput provided by the ILPs. While our greedy achieved 85%, 89%, and 91%, rnd_sieve fell behind with 85%, 81%, and 73% on the above graph instances, respectively (cf. [2]).

Main contributions. In this paper, *we provide a comprehensive algorithmic landscape of P4 pipeline embedding* both for the RMT and dRMT architectures. To the best of our knowledge, ours is the first principled approach to this end. We take off from a heavily simplified model of both programmable



(a) (b)

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

(c)

Fig. 2: The TDG and ODG 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) illustrates a straightforward RMT-embedding, (c) encodes an optimal dRMT-embedding of the program, where $P = 2$.

switch targets (the INF-CAP and BASIC models, respectively) on which pipeline embedding maps to well-known combinatorial optimization problems that can be solved to optimality in linear time. Then, *we gradually introduce additional degrees of complexity* to the bare-bones models to obtain increasingly more realistic and restrictive RMT and dRMT models, and *we give a comprehensive characterization of the respective computational complexity and approximation bounds*. As a main contribution, we show that the maximum throughput in these models is constant-approximable in linear time, but there is no Polynomial Time Approximation Scheme (PTAS) for any of the advanced models (unless $\mathcal{P}$=$\mathcal{NP}$). After presenting the formal problem definitions, the complexity landscape is summarized in Table IV.

For **RMT**, our results provide new characterizations for the P4 compilation/embedding efficiency obtainable with the techniques proposed in [17], [23]; see the results summarized in Table V. Importantly, we find that a simple application of the principles of the greedy pipeline embedding heuristics First Fit by Level (FFL) proposed in [17] and implemented in [23] to any of the RMT models studied in this paper already provides an approximation with a small constant gap, presenting an appealing choice when resource requirements are not that stringent. Our approximation algorithms may also be used to speed up optimal compilation, by providing provably good initial primal feasible solutions for bootstrapping the ILPs.

For the **dRMT** architecture, we determine the algorithmic complexity of different versions of the P4 program scheduling problem, and we present linear time constant-approximation algorithms. These results are summarized in Table VI. We provide lower bounds on the achievable throughput (or, equivalently, upper bounds on the achievable period P), a feature that has not been proposed in previous approaches.

The rest of the paper is organized as follows. In §II, we formalize the problem statement for each model and restate our main contributions formally. In §III, the linearly tractable simplified problem versions and related algorithms are discussed in more detail for both architectures. In §IV and §V, we build sequences of increasingly complex models to characterize the resource requirement of realizing P4 programs for RMT and dRMT, respectively. 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. Finally, in §VI and VII, we cast some open questions, and conclude the paper, respectively. In the main part, we focus on the claims and present only intuitive sketches of the proofs; the details are relegated to the Appendix.

II. Models and Main Results

The real complexity of P4 program 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 a 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 (for RMT) or clock cycles (for dRMT) in the switch pipeline. For instance, in RMT, a packet header field cannot be processed until a previous stage has finished modifying it. Similarly, in dRMT, the processor cannot start a given task before all the preceding tasks have finished. 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 amount of ternary match entries and only of limited width.

A. P4 program representations and example embeddings

To illustrate the P4 program embedding problem both to RMT and dRMT switches along with their similarities and differences, we use a toy program (taken from [22]). As depicted in Fig. 2a, the intermediate graph representation of the control-flow dependencies used by the compilers is slightly different in the studied architectures.

RMT switches use a so-called Table Dependency Graph (TDG), which is a directed, acyclic graph (DAG) of the dependencies (edges) between the logical match-action tables (MATs, considered as vertices) in the control flow. Dependencies arise between MATs that lie on a common path through the control flow, where table outcomes can affect the same packet. The dRMT architecture uses a somewhat finer representation of the P4 program called the Operation Dependency graph (ODG). The ODG is also a DAG. Intuitively, for larger flexibility, the ODG splits each MAT into a (possibly empty) match and action part. More formally, each node v_i of the TDG is represented in The ODG by match node m_i and action node a_i. There is an arc from m_i to a_i; all the in- and out-arcs of v_i are inherited by m_i and a_i, respectively. Finally, if m_i or a_i is empty (no match or action was indicated in v_i), we merge it to its twin by contracting arc (m_i, a_i).

In the example in Fig. 2a, we can see a simple TDG with a fork, having two leaf nodes v_1 and v_2 dependent on source v_0.

Notation	Meaning
$D_T = (V, E)$	Control-flow dependencies represented for RMT as a directed acyclic graph (DAG): Table Dependency Graph (TDG)
n_v	Height of node: max. # of entries in a node $v \in V$ ($n_v \in \mathbb{N}^+$)
n_S, n_T	Height (i.e., number of rows) in the SRAM and TCAM of a memory cluster ($n_S, n_T \in \mathbb{N}^+$)
w_v	Width of a node $v \in V$
w_T, w_S	Width (i.e., number of columns) of TCAMs, width of SRAMs
τ	Maximum number of tables per stage ($\tau \in \mathbb{N}^+$)
sram(v)	Denotes whether node v can be mapped to SRAMs (boolean)
$hsplit$	With $hsplit$ allowed, each node $v \in V$ can be (recursively) replaced in V by v_1, v_2 such that $n_{v_1} + n_{v_2} = n_v$, all the in-arcs and out-arcs of u are assigned to u_1 and u_2, resp., and an additional (u_1, u_2) arc is added to the arc-set.

TABLE II: Notations used in the Pipeline Embedding Problem (PEP) for the RMT architecture (Def. 1).

Notation	Meaning
$D_O = (V, E)$	Control-flow dependencies represented for dRMT as a (DAG): Operation Dependency Graph (ODG)
V_A, V_M	Sets of action and match nodes, resp. ($V = V_A \cup V_M$)
v, a, m	'node' ($v \in V$), 'action node' ($a \in V_A$), 'match node' ($m \in V_M$)
$l(v)$	CPU cycles to wait after the start of node v
ΔA, ΔM	For each action node a, $l(a) = \Delta A \in \mathbb{N}^+$; for each match node m, $l(a) = \Delta M \in \mathbb{N}^+$
$\overline{A}$, $\overline{M}$	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 ($\overline{A}, \overline{M} \in \mathbb{N}^+$)
w_m, w_a	Width of a match and action node ($w_m, v_a \in \mathbb{N}^+$)
IPC	*Inter-Packet Concurrency* (IPC): each processor in each cycle may start a match for at most a fixed number of IPC different packets and likewise start actions for up to IPC different packets

TABLE III: Notations used in the Disaggregated Pipeline Embedding Problem (DPEP) for the dRMT architecture (Def. 2).

The ODG representation is a straightforward transformation of the TDG, where m_0 being empty was merged back to a_0. We schedule these graphs to run both on RMT and dRMT, assuming both can perform up to 1 match every clock cycle in each stage/processor. For simplicity, we assume that there is no latency constraint (or, equivalently, every edge mandates a minimum latency of one cycle between the start of the operations on the edge).

The RMT pipeline requires a minimum of $S = 3$ stages as it lacks the necessary match capacity to simultaneously execute nodes v_1 and v_2 (containing matches m_1 and m_2), and both of these nodes must follow v_0 (see Fig. 2b). Assuming that there are $N = 2$ stages/processors (with the contents of the now non-existent stage 3 mapped to stage 1, and packets recirculated), by Eq. (1), this translates to a throughput of $t_{\text{RMT}}(2) = \min\{1, 1/\lceil 3/2 \rceil\} = 1/2$. Intuitively, this is because each packet has to traverse the pipeline twice.

In contrast, using these two processors, dRMT can schedule the same program to run at line rate (see Fig. 2b). Here, each row represents the sequence of operations for a single packet carried out on a single processor. It's worth noting that the period of this embedding is $P = 2$ (that is, each processor can terminate the processing of a packet in every second clock cycle), and packets are sent to processors in a round-robin fashion, where the packet that arrives in cycle k is processed by processor $k \mod 2$. The throughput is one packet per cycle, as can also be checked by Eq. (2): $t_{\text{dRMT}}(N) = \min\{1, N/P\} = \min\{1, 2/2\} = 1$.

B. Formal Models and Problem Statements

As indicated, from both RMT and dRMT, we will depart from polynomially solvable simplified problem definitions, then we will gradually add more levels of complexity to the problems in §IV and §V, respectively. However, for clarity, we already provide here the problem definitions of the most complex models tackled in this paper.

In the case of architecture RMT, based on Eq. 1, a program embedding that minimizes the number of stages S also maximizes the throughput. Thus, our aim will be minimizing S while respecting architectural and control-flow constraints. We call this problem the Pipeline Embedding Problem (PEP), and it is formally defined in Def. 1. In a nutshell, a PEP instance input consists of the following. A TDG, maximum number of entries in each node (table), SRAM, and TCAM heights (number of rows), node widths, TCAM/SRAM widths (number of columns), maximum number of tables per stage, and whether dividing a node consisting of an excessive amount of entries into separate nodes ($hsplit$) is allowed. The output is a mapping of TDG nodes to stages s.t. TDG arcs are forward, and memory constraints are satisfied. Note that for a node that has been $hsplit$-ted (maybe multiple times), only its starting and end stage is given as output. We formally define the problem as follows.

Definition 1. *We use the notations of Table II. The **Input** of a PEP instance consists of the following. A TDG $D_T = (V, E)$, height of node: max. # of entries in a node (table) n_v, height (i.e., number of rows) of SRAM, and TCAM (n_S and n_T, respectively ($+\infty$ if not considered)), node widths w_v (1 if not indicated), TCAM/SRAM widths w_T, w_S ($+\infty$ if not considered), max. # of tables per stage τ ($+\infty$ if not indicated), sram(v) denoting whether node v can be mapped to SRAMs for all $v \in V$ (*false *if only one resource considered), and $hsplit$ (only if indicated).*

*The **Output** is a mapping of TDG nodes to stages s.t. TDG arcs are forward, and resources of memory clusters are not exceeded. Sometimes we call this mapping a feasible embedding. Note that for a node that has been $hsplit$-ted (maybe multiple times) only its starting and end stage is given as output.*

*In a PEP instance, the **goal** is to find the **minimum number of stages** S for which a feasible embedding exists. In the decision version of PEP the input is extended with a value k, and the task is to decide if there exists a feasible embedding using at most $S = k$ stages.*

For dRMT, according to Eq. (2), a program embedding (scheduling) maximizes the throughput exactly when it minimizes the period P in which each processor can start/finish processing of a new packet. Thus, our aim is to minimize P. We call this problem the Disaggregated Pipeline Embedding

Model features (✓ = considered, ✗ = not considered)	*RMT models (PEP)*						*dRMT models (DPEP)*				
Problem denomination	INF-CAP	1D1R	1D1R-*hsplit*	2D1R	2D2R	2D2R-T/S	BASIC	IPC1	WIDTH	W-IPC1	W-IPC2
TDG/ODG dependencies	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
memory+processing disaggregation	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓
max. # entries in node, # rows of SRAM/TCAM (n_v, n_s, n_T)	✗	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗
hsplit- horizontal table splitting	✗	✗	✓	✓	✓	✓	(✓)	(✓)	(✓)	(✓)	(✓)
memory widths (w_S, w_T)	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗
memory types (TCAM/SRAM)	✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗
max. # tables per memory cluster (τ)	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
max. # parallel operations launchable ($\overline{A}, \overline{M}$)	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓
inter-packet concurrency (IPC)	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓	✓
Maximizing the throughput is part of complexity class... (unless $\mathcal{P} = \mathcal{NP}$)	$\mathcal{P}$	$\mathcal{APX} \setminus \mathcal{PTAS}$					$\mathcal{P}$	$\mathcal{APX} \setminus \mathcal{PTAS}$			

TABLE IV: Overview of results obtained in this paper. Throughput maximization, for our simplest problem settings, can be done in linear time for both RMT and dRMT. In the rest of the models, assuming $\mathcal{P} \neq \mathcal{NP}$, for a fixed (small) ϵ, the maximal throughput cannot be approximated in a factor of $1 - \epsilon$ in polynomial time (i.e., they are not part of the $\mathcal{PTAS}$ complexity class). As all our models allow polynomial-time approximation algorithms with approximation ratio bounded by a constant, they are part of the $\mathcal{APX}$ complexity class.

Problem, which is formally defined in Def. 2. We note that, partly because of the periodicity, to ensure a better formal administering of the dRMT throughput maximization, we raised the level of abstraction compared to the PEP problem definitions. In a nutshell, the input of a DPEP instance consists of the following. An ODG, sets of action and match nodes, latencies introduced by match and action nodes, the maximum number of parallel action field modifications and table searches per processor per cycle, widths of action and match nodes, and Inter Packet Concurrency (IPC) denoting the maximum number of packets for which each processor in each cycle may start a match or an action (these may be different). We formally define the problem as follows.

Definition 2. *We use the notations described in Table III. The **Input** of a DPEP instance consists of the following. An ODG $D_O = (V, E)$, sets of action and match nodes, resp. (V_A, V_M), latencies introduced by match and action nodes, resp. (ΔA, ΔM), maximum number of parallel action field modifications and table searches per processor per cycle $\overline{A}, \overline{M}$, width of action and match nodes w_a, w_m (1 if not indicated), Inter Packet Concurrency (IPC) value (1, 2, or, if not indicated, ∞).*

*The **Output** is a (DPEP) **scheduling** of the nodes, which is a function $\mathsf{S} : V \to \mathbb{N}^+$ such that for every arc $(v_i, v_j) \in E$ we have $\mathsf{S}(v_j) - \mathsf{S}(v_i) \geq l(v_i)$. For a scheduling S and period $P \in \mathbb{N}^+$, let S_P denote the set of schedulings S_i such that $\mathsf{S}_i(v) = \mathsf{S}(v) + iP$ (for $i \in \mathbb{N}$). We say that a scheduling S is **feasible with period** P if*

1) $\forall t \in \mathbb{N}^+ : \sum_{\mathsf{S}_i \in \mathsf{S}_P} \sum_{m \in V_M, \mathsf{S}_i(m) = t} w_m \leq \overline{M}$
2) $\forall t \in \mathbb{N}^+ : \sum_{\mathsf{S}_i \in \mathsf{S}_P} \sum_{a \in V_A, \mathsf{S}_i(a) = t} w_a \leq \overline{A}$
3) $\forall t \in \mathbb{N}^+ : \#\left\{\mathsf{S}_i \in \mathsf{S}_P \middle| \exists m \in V_M : \mathsf{S}_i(m) = t\right\} \leq IPC$
4) $\forall t \in \mathbb{N}^+ : \#\left\{\mathsf{S}_i \in \mathsf{S}_P \middle| \exists a \in V_A : \mathsf{S}_i(a) = t\right\} \leq IPC.$

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

Finally, in the complexity analysis, we use the unit cost arithmetic model of computation, where basic operations $+, -, *, /$ are unit-cost. Aligned to this, we suppose that n_v/n_S, n_v/n_T, w_S/w_v, w_T/w_v are $O(1)$. We argue these assumptions hold for practical settings, and unbounded ratios of the above fractions do not interfere with the polynomial solvability or constant approximability of the problems and add negligible time complexities that are straightforward to cope with.

C. Main results

The main results are summarized in Table IV. Columns of the Table correspond to PEP and DPEP problem versions, in increasing complexity from left to right. The rows of the Table correspond to different constraints, and marks ✓ and ✗ encode whether that specific constraint is taken into consideration for the specific problem. For example, TDG/ODG dependencies are taken into consideration for all the models, while memory and processing disaggregation is present only in the DPEP (dRMT models). We note that all the models will be detailed in the latter sections, and are specific cases of Definitions 1 and 2. For example, not considering the memory heights and widths, the maximum number of tables per memory cluster, or IPC can be translated to taking these numbers as being sufficiently large (or even infinite). Possible horizontal table splitting ($hsplit$) is only relevant in the PEP models, as in the DPEP problem formulations, conform to [22], we suppose memory clusters are sufficiently big (can host an infinite amount of entries). When we do not consider both memory types (TCAM/SRAM), we suppose there is a single type of memory that can host any node (e.g., TCAM). The high-level findings of this study can be summarized as follows.

MAIN RESULTS OF THIS PAPER. *In models INF-CAP and BASIC, the throughput can be maximized in linear time. In the rest of the models, the throughput is constant-approximable in linear time, but it is not approximable with arbitrary multiplicative precision in polynomial time, unless $\mathcal{P}$=$\mathcal{NP}$.*

The problems in INF-CAP and BASIC can be solved to optimality in linear time by claims formally proved in Appendix A. In all the other models, the throughput is constant approximable by combining Eq. (1) and (2) with theorems stating that the minimum stage number and the minimal period are constant-approximable, in the RMT and dRMT

architectures, respectively (by proofs in Appendix C). We get that in all the models different from INF-CAP and BASIC, the throughput is inapproximable with arbitrary multiplicative precision in polynomial time (unless $\mathcal{P}$= $\mathcal{NP}$) by combining Eq. (1) and (2) with theorems stating that the minimum stage number and the minimal period are $\mathcal{NP}$-hard to determine and to approximate (by proofs in Appendix B).

III. Tractable simplifications

A. RMT model INF-CAP: Mapping Concurrency

In this section, we consider a minimal problem underlying all pipeline embedding problems: the task is to correctly represent data- and control dependencies in a switch pipeline of infinite resources. We focus on the RMT architecture first. Recall that, for RMT, throughput maximization translates to minimizing the number of stages used.

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. 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 [17] for the details. In the TDG, 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 the objective is to minimize the number of stages used by the embedding.

As the simplest formulation of the P4 pipeline embedding problem for RMT (INF-CAP), below we require the MATs V to be embedded in the switch pipeline such that the arcs of the TDG are "forward"; i.e., for every $(l_a, l_b) \in E$, 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). We note that the algorithm for this model (Alg. 1) is the core of every approximation algorithm we give for more general models.

Results for this model. *Pipeline embedding under the INF-CAP model can be solved to optimality in linear time:* $O\big(|V| + |E|\big)$. *(by Claim 1)*

The proof is trivial: under INF-CAP one can obtain a correct embedding using topological sorting in polynomial time (in fact, linear) as follows. We group nodes that end the same length i of longest paths of dependencies into a so-called level $\mathcal{L}(i)$ according to Alg. 1. Then, we simply assign the nodes of each nonempty level $\mathcal{L}(i)$ to stage i. 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 [17]) 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 any compiler can routinely analyze (and optimize) such dependencies at scale.

Algorithm 1: Calculate levels

Input: DAG $D(V, E)$
begin
- $[v_1, v_2, \ldots, v_n] :=$ `TopologicalOrdering`$(D(V, E))$
- **for** $i = 1, \ldots, n$ **do**
  - $\mathcal{L}(i) :=$ empty list
  - **if** v_i *does not have any in-arc* **then**
    - $\lambda(v_i) := 1$
  - **else**
    - $\lambda(v_i) := \max\{\lambda(v_j) | (v_j, v_i) \in E\} + 1$
  - append v_i to $\mathcal{L}(\lambda(v_i))$
- **return** *Levels* $\mathcal{L}(1)$, $\mathcal{L}(2)$, $\ldots$, $\mathcal{L}(n)$

B. dRMT model BASIC

Recall that, in contrast to RMT, here, the throughput maximization translates to minimizing the period of P CPU cycles in which each processor can finish the processing of a packet. Also, to utilize the full potential of the disaggregations, here the P4 program is represented in a DAG slightly different from the TDG, as each node v of the TDG is split into a match and an action node m and a, with (m, a) added to the arcs of the resulting graph. This graph is called Operation Dependency Graph (ODG, [22]) $D_O = (V, E)$, $V = V_A \cup V_M$, where disjoint sets of vertices V_A and V_M represent the match and action nodes, 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. 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. E.g., 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 [22], we restrict our study to cyclic dRMT schedules, where a single packet processing plan is repeated on all packets processed by all processors (cf. [22, §3.2.]). To give an intuition behind our positive (approximability) results, [22, Theorem 3.5], the dRMT scheduling problem can be simplified to the problem of scheduling a single packet on a single processor. This single packet scheduling has to fulfill a requirement of P-*periodicity*: the set of nodes assigned to clock cycles t, $t + P$, $t + 2P$, $\ldots$ must meet the ΔM, ΔA, $\overline{M}$, $\overline{A}$ (and later on the width and inter-packet concurrency) requirements together, for all $t \in \{1, \ldots, P\}$.

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, where we sort the nodes according to an arbitrary topological order and place to the first available clock cycle (see Thm. 2).

RESULTS FOR THIS MODEL. *DPEP under the BASIC model can be solved to optimality in linear time:* $O\big(|E| + |V|\big)$. *(by Thm. 2)*

IV. (IN)APPROXIMABILITY IN RMT MODELS

A. 1D1R: Adding Simple Resource Constraints

Easily, INF-CAP yielded 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 RMT 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.

RESULTS FOR THIS MODEL. *Pipeline embedding under the 1D1R model is $\mathcal{NP}$-hard. Bad news: the optimal number of stages cannot be approximated better than $3/2$ unless $\mathcal{P}$=$\mathcal{NP}$. Good news: the optimum can be 3-approximated in linear time:* $O\big(|V| + |E|\big)$. *(by Lemma 3 & Claim 8)*

Intuitively speaking, the hardness and inapproximability results appear since 1D1R contains $\mathcal{NP}$-hard and inapproximable problems as special cases, like PARTITION [24]. Without going into details, in the construction used in Lemma 3, if the embedded $\mathcal{NP}$-hard problem instance has a solution, then we have an embedding in 2 stages, otherwise, we need at least 3 stages. The 3-approximability result can be stated using an algorithm similar to the one in the previous section solving INF-CAP (see Claim 8 in Appendix C). The resultant embedding is *guaranteed* to use at most 3 times the number of stages of the optimal one.

B. 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, [17] presents a technique to split MATs across multiple stages: one stores as many entries in the memory block of each of the upcoming stages as possible, until eventually all the entries of the MAT are assigned to a memory block. 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.

Algorithm 2: 1D1R-$hsplit$ Greedy Embedding

Input: TDG $D_T(V, E)$; stage size n_S; node sizes n_v
begin
- Calculate levels $\mathcal{L}(1)$, $\mathcal{L}(2)$, …, $\mathcal{L}(|V|)$ with Alg. 1
- $i := 1$
- **while** $\mathcal{L}(i) \neq \emptyset$ **do**
  - $\mathcal{E}_i := \mathcal{E}_{i-1} + \lceil (\sum_{v \in \mathcal{L}(i)} n_v)/n_S \rceil$
  - **for** $v \in \mathcal{L}(i)$ **do**
    - Reserve space for v continuously (using $hsplit$) in the next free spaces in stages $\mathcal{S}[\mathcal{E}_{i-1}+1], \ldots, \mathcal{S}[\mathcal{E}_i]$
  - $i := i + 1$
- **return** *Embedding*

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 $U \in V$ 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:

RESULTS FOR THIS MODEL. *Pipeline embedding under the 1D1R-hsplit model is $\mathcal{NP}$-hard. Bad news: the optimal number of stages cannot be approximated better than $5/4$ unless $\mathcal{P}$=$\mathcal{NP}$. Good news: the optimum can be 2-approximated in linear time:* $O(|V| + |E|)$. *(by Lemma 4 & Thm. 9)*

To obtain the approximation result, we designed Alg. 2. This algorithm is motivated by the greedy heuristics of [17]; accordingly, the approximability result can be applied to the generic greedy scheme under 1D1R-$hsplit$ with minor modifications. We partition the nodes into levels $\mathcal{L}(i)$ with Alg. 1. Nodes in a level are embedded consecutively into stages: denoting the j^{th} stage by $\mathcal{S}(j)$, if the last stage used by level $i-1$ is stage $\mathcal{S}(\mathcal{E}_{i-1})$, then nodes in level i are embedded into stages $\mathcal{S}(\mathcal{E}_{i-1}+1), \ldots, \mathcal{S}(\mathcal{E}_i)$ applying $hsplit$ if needed. We claim that Alg. 2 returns a correct embedding in linear time (see Thm. 9), and it is a 2-approximation. This is because the minimum number of stages needed for embedding the TDG (OPT) is at least the number h of used stages that are not full after running Alg. 2 since h is at most the length of the longest dependency chain in the TDG. Additionally, OPT is at least the number of stages fully utilized by Alg. 2.

The essence is that the additional degree of freedom introduced by $hsplit$ does not eliminate intractability. 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 another

RMT Model name	**INF-CAP**	**1D1R**	**1D1R-$hsplit$**	**2D1R**	**2D2R**	**2D2R-T/S**
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
Complexity	$\mathcal{P}$	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard	strongly $\mathcal{NP}$-hard
Bad news: Inapproximable better than... (unless $\mathcal{P}$=$\mathcal{NP}$)	OPT	$3/2 \cdot$ OPT	$5/4 \cdot$ OPT	$5/4 \cdot$ OPT	$5/4 \cdot$ OPT	$5/4 \cdot$ OPT
Good news: Constant approximable with...	OPT	$3 \cdot$ OPT	$2 \cdot$ OPT	$3 \cdot$ OPT	$(5 \text{ or } 8) \cdot$ OPT (*)	$(6 \text{ or } 9) \cdot$ OPT (*)

TABLE V: Main results for **RMT**. Bad news: the Pipeline Embedding Problem (PEP) is $\mathcal{NP}$-hard even with simple precedence and resource constraints. Good news: the minimum number of stages (and thus, the maximal throughput) in PEP is constant approximable in linear time, even for the most complex model. (*) means the lower bound holds in certain natural conditions explained in §IV-D.

way, the output of Alg. 2 can be used as a good initial primal feasible solution to bootstrap ILP problem formulations (like the one in [17]), 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 $5/4$).

C. 2D1R: Two-dimensional Resources

Up to this point, we have assumed that the stages have a certain number of rows available, each able to store a single table entry. In other words, each stage could store a predefined number of table entries, regardless of the widths of these entries. In reality, MATs have (at least) two size dimensions (number of rows as 'height', and header field match-width), and similarly, pipeline memory has two-dimensional capacities. Obviously, pipeline embedding should respect both dimensions: given the height n_v and the width w_v of each MAT $v \in V$, neither the number of rows used 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 and strip packing [25]. 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. Note that $hsplit$ is still permitted. More precisely, we may translate the optimization problem of model 2D1R to an abstract problem as follows. After possible $hsplit$ operations, each table is assigned to a stage such that precedence constraints are fulfilled. Tables in a stage are positioned such that 1) tables are non-rotatable rectangles that have parallel sides to the sides of stages (that are also considered rectangles), 2) rectangles of tables may be cut horizontally into smaller tables, but the number of cuts within a stage is bounded by $2 \cdot w_S/w_v$ for every node 3) no overlap between (the pieces of) tables is allowed.

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

RESULTS FOR THIS MODEL**.** *Pipeline embedding under the 2D1R model is $\mathcal{NP}$-hard. Bad news: the optimal number of stages cannot be approximated better than $5/4$ unless $\mathcal{P}$=$\mathcal{NP}$. Good news: the optimum can be 3-approximated in linear time: $O\big(|V| + |E|\big)$.* *(by Lemma 4 & Claim 10)*

Here, only 3-approximability needs more explanation: we again motivate this result with a (sketch of a) proof. The approximation algorithm for 2D1R is quite similar to Alg. 2 we used for 1D1R-$hsplit$, the only difference is in the packing of the levels. Initially, inside each level $\mathcal{L}(i)$, we partition nodes into groups according to their width. Without loss of generality, we scale the widths such that each stage is of width 1. First, MATs of width $\geq 1/2$ are packed (in one column), possibly applying $hsplit$, then MATs of width in $[1/4, 1/2)$ are packed in two columns, etc. Generally, let $V_{w,k}$ denote the set of MATs of length $[1/2^{k+1}, 1/2^k)$. After packing $V_{w,k-1}$ is finished, MATs in $V_{w,k}$ are packed in 2^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 pack $V_{w,k}$ such that we assign some of its elements to the remaining space of the row. One can show that with this modification Alg. 2 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 OPT stages that are not at least halfway full after Alg. 2 terminates (see Claim 10 in Appendix C for details).

D. 2D2R: Both SRAMs and TCAMs Available

In most switch ASICs, there are multiple types of memory, each optimized for different purposes [10]: 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 for storing 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 more general model than 2D1R, 2D2R is also $\mathcal{NP}$-hard and inapproximable below $5/4$ (unless $\mathcal{P}$=$\mathcal{NP}$, see Lemma 4 for a formal proof). On the positive side, in Appendix C, we show a modification of Alg. 2 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 in the next free stages. In all the previous three phases, embedding follows the steps of the

2D1R version of Alg. 2, 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 11. Hence, our results establish appealing approximation bounds of greedy heuristics under 2D2R as well:

RESULTS FOR THIS MODEL. *Pipeline embedding under the 2D2R model is $\mathcal{NP}$-hard. Bad news: the optimal number of stages cannot be approximated better than $5/4$ unless $\mathcal{P}$=$\mathcal{NP}$. Good news: the optimum can be 8-approximated in linear time: $O\big(|V|+|E|\big)$. If $w_T \geq w_S$, it can be 5-approximated in the same complexity.* *(by Lemma 4 & Claim 11)*

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

In recent programmable switch ASICs [10] the number τ 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 $\mathcal{NP}$-hard.

RESULTS FOR THIS MODEL. *Pipeline embedding under 2D2R-T/S is strongly $\mathcal{NP}$-hard and inapproximable better than $5/4$ unless $\mathcal{P}$=$\mathcal{NP}$. Good news: the optimum can be 9-approximated in linear time: $O(|V|+|E|)$. If $w_T \geq w_S$, it can be 6-approximated in the same complexity.* *(by Lemma 4, Claim 5, & Claim 12)*

The complexity of the problem can be seen through a reduction from the strongly $\mathcal{NP}$-hard `3-PARTITION` problem [24, SP15]. The proof of the approximability can be made using the greedy heuristic scheme of model 2D2R, with the modification that wherever the number of MATs assigned to the current stage would exceed the allowed maximum τ, we start a new stage. Then, one can show that there are only OPT more stages needed for embedding under 2D2R-T/S than under 2D2R since the number of stages hosting exactly τ MATs is $\leq$OPT.

F. 2D2R-PISA: Fully-fledged PISA Model

The constraints incurred by a real switch go beyond our simplified models; see [17], [23]. 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 $\mathcal{NP}$-hard). The design and formal analysis an efficient approximation for this model would exceed the limits of this paper. However, we note that our algorithms presented for the RMT models of this paper are slight variants of the First Fit by Level (FFL) heuristic of [17], which was used as the greedy algorithm on Table I. Thus, the gap of a number of 8% to 15% extra stages used by the FFL compared to the ILP optimum gives a convincing hint of the practical effectiveness of our approximation algorithms for the RMT models.

V. (IN)APPROXIMABILITY IN DRMT MODELS

Our next goal is to rebuild step-by-step, and then to analyze the dRMT models of [22]. To this end, some of the memory constraints tackled in the case of the RMT architecture will not be considered here. We argue that, intuitively, introducing the same memory constraints as in the RMT models still enables simple constant-factor approximation algorithms similar to those discussed throughout this study.

A. 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 Thm. 6). On the bright side, in this setting, there exists a 3-approximation algorithm, described in Thm. 16.

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: $O\big(|V|+|E|\big)$.* *(by Thm. 6 & 16)*

B. WIDTH: Variable table widths

Next, on top of BASIC we will also allow each match and action node m and a to be of arbitrary width, measured by a positive integer w_m and v_a, respectively. 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 Thm. 7):

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 $\mathcal{P}$=$\mathcal{NP}$. Good news: the optimum can be $3/2$-approximated in linear time: $O\big(|V|+|E|\big)$.* *(by Thm. 7 & 13)*

We note that in the case of WIDTH, the absolute inapproximability and approximability ratios provided in our results are the same. Thus, these ratios are tight. This phenomenon stems from the fact that this model is very closely related to the bin-packing problem. More concretely, for both the match and action nodes, one can assign a congruence class to each bin in a solution of a bin-packing problem instance with the nodes, their widths, and $\overline{M}$, or $\overline{A}$ as input parameters, respectively. Thus, a WIDTH problem instance can be solved via a postprocessing of a bin-packing problem solution. It is known that the bin-packing problem can be solved in linear time with an absolute approximation ratio of $3/2$ [26], and that this ratio is the best, unless $\mathcal{P}$=$\mathcal{NP}$ [27]. The $3/2-\epsilon$

Model name:	BASIC	IPC1	WIDTH	W-IPC1	W-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 (≤2 pkt./proc./cycle)
Complexity	$\mathcal{P}$	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard
Bad news: Inapproximable better than ... *(,unless $\mathcal{P}$=$\mathcal{NP}$)*	OPT	$4/3 \cdot$ OPT	$3/2 \cdot$ OPT	$3/2 \cdot$ OPT	$3/2 \cdot$ OPT
Good news: Constant approximable in...	OPT	$3 \cdot$ OPT	$3/2 \cdot$ OPT	$4 \cdot$ OPT	$8 \cdot$ OPT

TABLE VI: Overview of the main results for **dRMT**. Bad news: the Disaggregated Pipeline Embedding Problem (DPEP) is $\mathcal{NP}$-hard even with relaxing some constraints. Good news: the DPEP is polynomially solvable under the BASIC model, and the minimum period of a scheduling (thus also the throughput) is constant-approximable in linear time even when considering the model tackled by [22].

inapproximability result of the bin-packing is also a corollary of the $\mathcal{NP}$-hardness of the PARTITION problem [27].

C. W-IPC1: Main dRMT model

Our next model, W-IPC1, is equivalent to the one studied in [22]. 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 Thm. 6). As a positive result, we show that in W-IPC1, the optimum can be 4-approximated in linear time; see Alg. 4, and Thm. 15.

The algorithm is based on the observation that the optimal period for a scheduling solution (see Def. 2) 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 14). Our approach, at first, greedily finds a solution with $\Delta M = \Delta A = 1$ (a *simplified scheduling*, see Def. 3) such that clock cycles are filled with match/action nodes at least half full when possible. Then, this simplified scheduling is expanded to a proper scheduling in Alg. 4. A feature of our solutions is that no match and action nodes are assigned to the same clock cycle.

RESULTS FOR THIS MODEL. *DPEP under the W-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 linear time: $O(|V| + |E|)$.* *(by Thm. 7 & 15)*

We note that, in model W-IPC1, [2] found that a variation of Alg. 4 achieves at least 85% of best throughput achievable by ILP formulations on all P4 programs studied in [22], significantly improving on the heuristic rnd_sieve of [22] that achieves only 73% of the best ILP solution.

D. W-IPC2: Loose IPC constraints

The original paper [22] also considers the case when IPC is 2, possibly allowing more compact program embeddings. Intuitively, increasing IPC from 1 to 2 may allow ≤ 2 as efficient embeddings. Thus, the greedy algorithm of model W-IPC1 is 8-approximation in model W-IPC2 (see Thm. 17).

RESULTS FOR THIS MODEL. *DPEP under the W-IPC2 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 8-approximated in linear time: $O(|V| + |E|)$.* *(by Thm. 7 & 17)*

For IPC= 2, the ILP solvers of [22] can compute efficient program embeddings relatively easily. Thus, we will not study this model further here.

Algorithm 3: Simplified Scheduling

Input: ODG $D_O = (V = V_M \cup V_A, E)$; $W : V \to \mathbb{N}^+; \overline{M}, \overline{A}$
Output: Simplified scheduling $\sigma : V \to \mathbb{N}^+$
begin
1 Calculate levels $\mathcal{L}(1)$, $\mathcal{L}(2)$, ..., $\mathcal{L}(|V|)$ with Alg. 1
2 values of simplified scheduling σ: not yet defined
3 $i := 1$
4 **while** $\mathcal{L}(i) \neq \emptyset$ **do**
5 Embed $\mathcal{L}(i) \cap V_M$ to the next free clock cycles of σ with the algorithm of [26]
6 Embed $\mathcal{L}(i) \cap V_A$ to the next free clock cycles of σ with the algorithm of [26]
7 $i := i + 1$
return Simplified scheduling σ

Algorithm 4: W-IPC1 Greedy Scheduling

Input:
ODG D_O=(V=$V_M \cup V_A, E$); W:$V \to \mathbb{N}^+; \overline{M}, \overline{A}$, ΔM, ΔA
Output: Scheduling $\mathsf{S} : V \to \mathbb{N}^+$
begin
1 Calculate simplified scheduling σ with Alg. 3
2 $f(0) := 0$, $p_m := 1$, $p_a := 1$
3 **for** $i = 1, \ldots,$ *length*(σ) **do**
if *there is a node in $\sigma(i)$ having an in-arc* **then**
$\delta(i) := \max\{f(\sigma(v)) + l(v))|(v, w) \in E, \sigma(w) = i\}$
else
$\delta(i) := f(i-1) + 1$
if $\sigma^{-1}(i) \subseteq V_M$ **then**
$f(i) := \min\{k \geq \delta(i)|\ k \equiv p_m \mod P\}$
4 $p_m := p_m + 1$
else
$f(i) := \min\{k \geq \delta(i)|\ k \equiv p_a \mod P\}$
5 $p_a := p_a + 1$
6 **return** *Scheduling* $\mathsf{S} := f \circ \sigma$

VI. Discussion

This study aimed to discuss the most important algorithmic questions left open in RMT and dRMT base papers [17] and [22], respectively. Due to space issues, some constraints were inevitable to be left out of our investigation, just as [17] and [22] did in their own case. One of these relaxations is that this study did not model storage constraints for the parsed packet headers and metadata. However, we believe our approximation algorithms can be extended to cope with these constraints, too, using similar techniques as those presented in the Appendix (note that these constraints are also only tangentially addressed in [17], [22]).

Further, at the RMT models, in line with both [17] and [22], our optimization objective was to minimize the number S of stages needed to embed the P4 program. This, by Eq. 1 (also cf. [22, Eq. (9)]), in terms of throughput maximization, is just a more fine-grained version of a bit more relaxed objective of *minimizing the number of packet recirculations*. We note that with this alternative relaxed optimization objective, all of our complexity findings for RMT models remain valid. Namely, *in model INF-CAP, the throughput can be maximized in linear time, while in the rest of the models, the throughput is constant-approximable in linear time, but it is not approximable with arbitrary multiplicative precision in polynomial time, unless $\mathcal{P}$=$\mathcal{NP}$.*

Lastly, remaining at the possibility of packet recirculations on an RMT switch, we can see that the stages in each $\mod N$ stage class share the same memory, where N denotes the number of stages. This constraint is similar to those seen in our dRMT models. Our certitude is that our approximation techniques presented in the Appendix carry over if factoring in cyclic memory constraints for RMT. We left this constraint relaxed, because, in this study, our principle was to answer the most urgent algorithmic questions left open by [17], [22], and not starting from scratch in our own way.

Furthering our findings by factoring in additional constraints or by considering different objective functions may be a straightforward target of follow-up studies.

VII. Conclusion

This paper investigates the algorithmic aspects of P4 pipeline embedding on a series of relaxed RMT and dRMT models, with a focus on maximizing switch throughput. We report mixed findings: while P4 embedding problems, in general, are $\mathcal{NP}$-hard even in severely relaxed formulations, *the maximal throughput is constant-approximable in linear time* even in the most complex pipeline models. Evaluations show that our algorithms can rapidly compute high-quality heuristic embeddings for real-life P4 programs. Our algorithms may also be used to speed up optimal compilation, by providing provably good initial feasible solutions for the ILP solver running inside commercial P4 compilers. Our study reinforces the earlier finding [1], [2] that, despite its inherent complexity, P4 compilation is efficiently approximable, in that it is possible to obtain *provably good embeddings in provably short time*.

References

[1] B. Vass, E. Bérczi-Kovács, C. Raiciu, and G. Rétvári, "Compiling packet programs to reconfigurable switches: Theory and algorithms," in *Proceedings of the 3rd P4 Workshop in Europe*. ACM, 2020.

[2] B. Vass, A. Fraknói, E. Bérczi-Kovács, and G. Rétvári, "Compiling packet programs to drmt switches: Theory and algorithms," in *Proceedings of the 5th P4 Workshop in Europe*. ACM, 2022.

[3] N. McKeown, "Programmable forwarding planes are here to stay," in *ACM SIGCOMM NetPL*, 2017.

[4] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM CCR*, vol. 44, pp. 87–95, 2014.

[5] H. T. Dang, M. Canini, F. Pedone, and R. Soulé, "Paxos made switch-y," vol. 55, no. 3. ACM SIGCOMM, 2016, pp. 19–24.

[6] C. Kim, A. Sivaraman, N. P. Katta, A. Bas, A. Dixit *et al.*, "In-band network telemetry via programmable dataplanes," 2015.

[7] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "Hula: Scalable load balancing using programmable data planes," in *ACM SOSR*, 2016, pp. 1–12.

[8] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "Silkroad: Making stateful Layer-4 load balancing fast and cheap using switching ASICs," in *ACM SIGCOMM*, 2017, pp. 15–28.

[9] "Switch.p4," https://github.com/p4lang/switch/blob/master/p4src/switch.p4, accessed: 2020-08-19.

[10] Barefoot, "Intel/Barefoot Tofino 2: Product Brief," https://www.barefootnetworks.com/products/brief-tofino-2, accessed: 2020-09.

[11] T. P. L. Consortium, "Behavioral model (bmv2)," https://github.com/p4lang/behavioral-model, accessed: 2022-09.

[12] W. Tu, F. Ruffy, and M. Budiu, "P4c-xdp: Programming the linux kernel forwarding plane using p4," in *Linux Plumbers Conference*, 2018.

[13] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh *et al.*, "Packet transactions: High-level programming for line-rate switches," in *ACM SIGCOMM*, 2016, pp. 15–28.

[14] J. Gao, E. Zhai, H. H. Liu, R. Miao, Y. Zhou *et al.*, "Lyra: A cross-platform language and compiler for data plane programming on heterogeneous ASICs," in *ACM SIGCOMM*, 2020, p. 435–450.

[15] P. G. Patra, C. E. Rothenberg, and G. Pongracz, "Macsad: High performance dataplane applications on the move," in *IEEE HPSR*, 2017.

[16] H. Wang, R. Soulé, H. T. Dang, K. S. Lee, V. Shrivastav *et al.*, "P4fpga: A rapid prototyping framework for p4," in *Proceedings of the Symposium on SDN Research*, 2017, pp. 122–135.

[17] L. Jose, L. Yan, G. Varghese, and N. McKeown, "Compiling packet programs to reconfigurable switches," in *USENIX NSDI*, 2015.

[18] "Netronome programmer studio 6.0 software development kit," https://opennetworking.org/wp-content/uploads/2021/05/2021-P4-WS-Vladimir-Gurevich-Slides.pdf, accessed: 2022-09.

[19] Intel Corporation, "Intel p4 studio," https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/p4-suite/p4-studio.html, accessed: 2022-09.

[20] X. Gao, T. Kim, M. D. Wong, D. Raghunathan, A. K. Varma *et al.*, "Switch code generation using program synthesis," in *ACM SIGCOMM*, 2020.

[21] H. Soni, M. Rifai, P. Kumar, R. Doenges, and N. Foster, "Composing dataplane programs with μP4," in *ACM SIGCOMM*, 2020, p. 329–343.

[22] S. Chole, A. Fingerhut, S. Ma, A. Sivaraman, S. Vargaftik *et al.*, "dRMT: Disaggregated Programmable Switching," in *ACM SIGCOMM*, 2017.

[23] "Bitbucket code repository of [17]," https://bitbucket.org/lavanyaj/switch-compiler/src/master/, accessed: 2020-08-19.

[24] M. R. Garey and D. S. Johnson, "Computers and intractability: A guide to the theory of NP-completeness," 1978.

[25] H. I. Christensen, A. Khan, S. Pokutta, and P. Tetali, "Approximation and online algorithms for multidimensional bin packing: A survey," *Computer Science Review*, vol. 24, pp. 63–79, 2017.

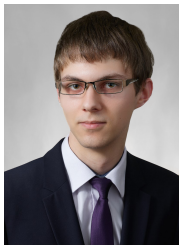
[26] R. Berghammer and F. Reuter, "A linear approximation algorithm for bin packing with absolute approximation factor 32," *Science of Computer Programming*, vol. 48, pp. 67–80, 2003.

[27] D. Simchi-Levi, "New worst-case results for the bin-packing problem," *Naval Research Logistics*, vol. 41, pp. 579–585, 1994.

[28] J. K. Lenstra and A. Rinnooy Kan, "Complexity of scheduling under precedence constraints," *Operations Research*, vol. 26, pp. 22–35, 1978.

Balázs Vass received his MSc degree in applied mathematics at ELTE, Budapest in 2016. He finished his PhD in informatics in 2022 on the Budapest University of Technology and Economics (BME). His research interests include networking, survivability, combinatorial optimization, and graph theory. He was an invited speaker of COST RECODIS Training School on Design of Disaster-resilient Communication Networks '19. He is a TPC member of IEEE INFOCOM '23, '24, and '25.

Erika R. Bérczi-Kovács received the M.Sc. Degree in Mathematics and the Ph.D. degree in Applied Mathematics from the Eötvös Loránd University (ELTE), Budapest, in 2007 and 2015, respectively. She is currently a senior lecturer at the Department of Operations Research, ELTE, and she is with the MTA-ELTE Egerváry Research Group on Combinatorial Optimization. Her research interests are combinatorial optimization and operations research. She was a recipient of NaNa 2016 Best Paper Award.



Ádám Fraknói received his BSc in mathematics from the Eötvös Loránd University (ELTE), Budapest, Hungary. Now he is a master's student also at ELTE mathematics. He has acquired first prizes at the university-wide and nationwide Scientific Students Association Conferences, as well as various scholarships. He is interested in theoretical computer science and machine intelligence.

Costin Raiciu received his B.Sc. and M.Sc. from University Politehnica of Bucharest in 2003 and 2004, and his PhD from University College London in 2011. Costin is now Professor in the Computer Science Department of University Politehnica of Bucharest. His research interests include networking, systems and verification. Costin is keen on pushing his research work into production, with Multipath TCP, and EQDS example protocols undergoing standardization or already deployed.

Gábor Rétvári received the M.Sc. and Ph.D. degrees in electrical engineering from the Budapest University of Technology and Economics in 1999 and 2007, where he is now a Senior Research Fellow. His research interests include all aspects of network routing and switching, the programmable data plane, and the networking aspects of cloud native computing. He is a co-founder and CTO of L7mp Technologies, a company specializing in running large-scale WebRTC services over Kubernetes.

APPENDIX

A. Linear time algorithms for optimal throughput

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

Proof: First, we add a hypothetical MAT L to the TDG with arcs (L, l_i) to all source nodes of the TDG. Then, for each j, embed all MATs of level $\mathcal{L}(j)$ stage j. It is easy to see that this algorithm returns a valid TDG embedding with the minimal theoretically possible stages, that is, the vertex-length of the longest directed path in the TDG. ∎

Theorem 2. *For model BASIC,* $P = \max\left(\left\lceil \frac{|V_M|}{M} \right\rceil, \left\lceil \frac{|V_A|}{A} \right\rceil\right)$ *is the optimal period, and a feasible* P*-periodic scheduling can be found in linear time:* $O\big(|E|+|V|\big)$.

Proof: It is clear that $\lceil |V_M|/M \rceil$ and $\lceil |V_A|/A \rceil$ 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, filling up the modulo classes one by one, starting from 1 to P. We denote the number of match and action nodes already scheduled $\#_m$ and $\#_a$, respectively. $\mathsf{S}(v_1) := 1$. For $j > 1$, let $\delta_j := \max\{\mathsf{S}(v_i) + l(v_i) \mid v_i v_j \in E\}$ if v_j has at least one entering arc, otherwise $\delta_j := \mathsf{S}(v_{j-1})$. If $v_j \in V_M$, let $\mathsf{S}(v_j) := \min\{k \geq \delta_j \mid k \equiv \lfloor \#_m/M \rfloor\}$. Similarly, if $v_j \in V_A$, let $\mathsf{S}(v_j) := \min\{k \geq \delta_j \mid k \equiv \lfloor \#_a/A \rfloor\}$. Note that by choice of P, we do not overbook any clock cycle. The total number of steps for calculating all δ_i values is $O(|E|)$. Then, determining values $\mathsf{S}(i)$ and maintaining counters $\#_m$ and $\#_a$ take $O(|V|)$ time, that gives a total running time of $O\big(|E|+|V|\big)$. ∎

B. $\mathcal{NP}$-Hardness and inapproximability proofs

In **RMT models**, our aim is to find an embedding of the given TDG that uses the least stages possible. For the $\mathcal{NP}$-hardness and inapproximability results we consider the *decision* version of these problems, which asks if a given number of k stages given in the input is enough to embed a given TDG (or alternatively, if a TDG can be mapped on a given switch architecture). Clearly, the decision version of every investigated model is in $\mathcal{NP}$.

Lemma 3. *The decision version of RMT model 1D1R is* $\mathcal{NP}$*-complete, and the optimum cannot be approximated better than a ratio of* $3/2$*, unless* $\mathcal{P}$=$\mathcal{NP}$.

Proof: We reduce our problem to the $\mathcal{NP}$-complete `PARTITION` [24], where the task is to decide whether a multiset of positive integers $a_1, \ldots, a_l$ of sum $2K$ can be partitioned into two sets of sums K. Given a `PARTITION` instance, we consider the 1D1R instance where for each value a_i a node v_i is created with node size $n_{v_i} = a_i$. No arcs are added to the TDG graph. Let the size of the stages be K. The given `PARTITION` instance has a solution if and only if the corresponding 1D1R TDG can be embedded in 2 stages, which proves $\mathcal{NP}$-completeness for $k = 2$. The inapproximation ratio $3/2$ follows from the $\mathcal{NP}$-completeness of case $k = 2$. ∎

Note that the ratio above holds both for small and large optimal stage numbers, since one can take sequentially each after multiple (different) instances as the one described in the proof to create a compound instance of arbitrarily large size.

Lemma 4. *The decision versions of RMT models 1D1R-hsplit, 2D1R, 2D2R, and 2D2R-T/S are* $\mathcal{NP}$*-complete, and the optimum cannot be approximated better than a ratio of* $5/4$*, unless* $\mathcal{P}$=$\mathcal{NP}$.

Proof: It is enough to prove the $\mathcal{NP}$-completeness and inapproximability for model 1D1R-*hsplit*, since it is a special case of both 2D1R, 2D2R, and 2D2R-T/S. We will prove its NP-completeness by reducing an instance of `EQUAL CARDINALITY PARTITION` (`ECP`) problem, which is known to be $\mathcal{NP}$-complete [24, *version of* SP12]. We note that a visual representation of the upcoming construction

Minimizing the #stages (N) or period (P) is...	Problem denomination	INF-CAP	1D1R	1D1R-*hsplit*	2D1R	2D2R	2D2R-T/S	BASIC	IPC1	WIDTH	W-IPC1	W-IPC2
	solvable in linear time	Claim 1			✗			Thm. 2			✗	
	$\mathcal{NP}$-hard and inapproximable in polynomial time (unless $\mathcal{P}$=$\mathcal{NP}$)	✗	Lemma 3		Lemma 4			✗	Thm. 6		Thm. 7	
	constant approximable in linear time	(Claim 1)	Claim 8	Thm. 9	Claim 10	Claim 11	Claim 12	(Thm. 2)	Thm. 16	Thm. 13	Thm. 15	Thm. 17

TABLE VII: The formal proofs of our different propositions can be found in the above-mentioned Claims/Lemmas/Theorems.

used in the proof can be found in [1, Fig. 2]. Let an ECP instance be given: positive integer values $a_1, \ldots, a_{2k}$ such that $\sum_{i=1}^{2k} a_i = 2K$. The task is to decide whether there is a k-size subset I of the values such that $\sum_{i \in I} a_i = K$.

We construct a corresponding 1D1R-*hsplit* problem. Consider a TDG with 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 having k, $K+k$, $K+k$ and k rows, respectively. Each $\{(f_i, s_i), (s_i, l_i)\} \subseteq E$. Node demands are $n_{s_i} = a_i$ and $n_{f_i} \equiv n_{l_i} \equiv 1$, so $\sum_{i \in \{1,\ldots 2k\}} n_{s_i} = 2K$. Due to that the total sum of stage sizes equals the total sum of node demands, *hsplit* cannot be used in a valid embedding, since it would cause not to fully utilize the capacities of a stage from among stages $1, \ldots, 4$, and some fragments of some tables could not be embedded in any stage. More precisely, the only way these tables can fit in the 4 stages is if there exists a set of indexes $I \subset \{1, \ldots, 2k\}$, $|I| = k$ 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 $j \in \{1, \ldots, 2k\} \setminus I$, f_j assigned to stage 2, s_j are in stage 3, l_i are in stage 4, and $\sum_{i \in I} n_{s_i} = K$. Thus a valid embedding into four stages exists exactly if there exists such an I ($|I| = k$) for which $\sum_{i \in I} n_{s_i} = K$. Next, we extend the above setting to get a proper PEP instance with equal stage capacities for which the above reasoning still holds. To this end, we modify stage 1 and stage 4 to have $K + k$ rows (just as stages 2 and 3). Further, we add to the TDG nodes F and L, both having a height of K. Finally, for each s_i, (F, s_i) and (s_i, L) is added to E (see also Fig. 2 in [1]). Since in the above setting, the TDG can be embedded in 4 stages exactly if the embedded ECP problem instance has a solution, the $\mathcal{NP}$-completeness of the problem follows. The inapproximation ratio $5/4$ follows from the $\mathcal{NP}$-completeness of case $k = 4$. ∎

Claim 5. *The decision version of pipeline embedding under 2D2R-T/S is strongly $\mathcal{NP}$-complete.*

Proof: 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 widths (i.e., for all $v \in V$, $w_v = w_S = w_T$), thus the second size dimension can be ignored. Let all the MATs be assignable to any of the resources (that is, sram(v) is true for every node v), thus we can characterize the size of each stage with only the number of its rows, and maximum number of tables that can be assigned to the stage, that are $n_S + n_T$ and τ, respectively. Let the pipeline have m stages of size $B \in \mathbb{Z}^+$ ($B = n_S + n_T$), each stage capable of storing at most $\tau = 3$ tables. Let the TDG to be embedded consist of $3m$ MATs of size $B/4 < n_v < B/2$ (for every $v \in V$) without any TDG dependencies. The sizes of the nodes add up to mB. Then, the decision of whether the TDG can be embedded into the pipeline is equivalent to the 3-PARTITION problem, which is known to be strongly $\mathcal{NP}$-complete [24, SP15]. ∎

Now we turn to the complexity of the **dRMT models**.

Theorem 6. *The decision version of model IPC1 is $\mathcal{NP}$-complete. Furthermore, the optimal period cannot be approximated better than a ratio of $4/3$ unless $\mathcal{P}$=$\mathcal{NP}$.*

Proof: Consider an instance of the scheduling problem $P|prec, p_j = 1|C_{max}$. It is $\mathcal{NP}$-complete to decide whether $C_{max} > 3$ for this problem [28]. We construct a corresponding IPC1 instance the following way: nodes correspond to jobs (all nodes are action nodes), the ODG graph is the graph of precedence constraints with latency constant 1 and $\overline{A}$ is the number of machines. Then the optimal P is 3 if and only if $C_{max} = 3$ (see Lemma 14). ∎

Theorem 7. *The decision versions of models WIDTH, W-IPC1 and W-IPC2 are all $\mathcal{NP}$-complete, and the optimal periods cannot be approximated better than a ratio of $3/2$, unless $\mathcal{P}$=$\mathcal{NP}$.*

Proof: We use a reduction from the $\mathcal{NP}$-complete PARTITION PROBLEM [24]. Let a multiset Z of positive integers $z_1, \ldots, z_n$ be given, where $\sum_{i=1}^{n} z_i = 2K$. We create a model instance such that the optimal period P is 2 if and only if Z can be partitioned into two subsets of equal sum, independent of the value of IPC: consider an instance with n action nodes $a_1, \ldots, a_n$ and empty control-flow dependencies (i.e. $E = \emptyset$). Let $w_{a_i} := z_i$ and $\overline{A} := K$, and let P denote the optimal period. Then $P \geq \sum_{i=1}^{n} w_{a_i}/\overline{A} = 2K/K = 2$. On one hand, if Z has an equal partition into two subsets, then clearly $P = 2$ (for every IPC value). On the other hand, if $P = 2$, then there is a value F such that each clock cycle i is full with respect to $\overline{A} = K$ if $i \geq F$. The scheduling induces a partition $\mathcal{P}$ of V into sets $\mathcal{P} = \{R_1, \ldots, R_k\}$ representing nodes of a package in the same clock cycle. Then consider subset I of $\mathcal{P}$ containing all sets R_i scheduled for clock cycle F (possibly from different packets): $\sum_{i \in I} \sum_{a_i \in R_i} w_{a_i} = K$. Then I and $\mathcal{P} \setminus I$ induce an equal partition of Z indeed. ∎

Corollary 1. *For any given value k of the Inter Packet Concurrency, the decision version of a DPEP model W-IPC-k is $\mathcal{NP}$-complete and the optimal period cannot be approximated better than a ratio of $3/2$, unless $\mathcal{P}$=$\mathcal{NP}$.*

C. Linear time constant approximation algorithms

For **RMT models**, the basic idea behind our algorithms and proofs of constant approximations is the following. The TDG nodes are grouped in sets $\mathcal{L}(1), \mathcal{L}(2), \ldots,$ (called levels) 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 models, we prove that, among the stages used in our embedding, with at most $c_1 \cdot \text{OPT}$ 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 for the embedding. That is, 1) in case of PEP models 1D1R, 1D1R-$hsplit$, and 2D1R, for each table, we only store its starting stage and position, and its end stage and position, respectively; and 2) in case of models 2D2R, and 2D2R-T/S we store these values for each table for both kind of resources (TCAM and SRAM). Clearly, the actual embedding phase needs some additional time. Since our outputs are constant approximations of the optimal stage number OPT, this additional time scales linearly not only with the stage sizes, but with OPT too.

Claim 8. *The optimum of 1D1R can be* 3*-approximated in linear time:* $O\big(|V| + |E|\big)$.

Proof: We partition the nodes into levels as in Alg.1 for INF-CAP, where level $\mathcal{L}(i)$ contains the set of nodes that have a longest ending path containing i nodes. 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 $\mathcal{L}(j)$. Using the linear-time algorithm of [26], we calculate a ($3/2$-approximate) solution for the bin packing problem initialized with the stage size and MAT sizes of level $\mathcal{L}(j)$. We assign one of the next available stages for each of these bins. We claim that this gives a valid embedding. The output uses at most 3·OPT stages since 1) the stages of each level are at least half-full, except for the last stage, and 2) the last stages of the levels are at most OPT.

Speaking of the complexity, all of computing the levels, algorithm of [26], and all the rest of computations take $O\big(|V| + |E|\big)$ time, completing the proof. ∎

Theorem 9. *Alg. 2 gives a 2-approximation for the pipeline embedding problem under 1D1R-*$hsplit$ *in linear time:* $O(|V| + |E|)$.

Proof: Correctness: For any $u, v \in V$, (u, v) being a dependency arc means $\lambda(v) > \lambda(u)$ in the output, where $\lambda(w)$ denotes the stage table w is embedded. Thus, Alg. 2 returns a correct embedding. 2*-approximation:* In every level i only the last stage $\mathcal{S}(\mathcal{E}_i)$ can be not full. The number of these last stages is the length of the longest path in the graph, which is at least OPT. On the other hand, OPT is at least the number of stages fully utilized by Alg. 2. Consequently, it uses at most $2 \cdot \text{OPT}$ stages. *Complexity:* At line 1, we return a topological ordering, that can be done in $O(|V|+|E|)$. Since the arithmetic operations have constant cost in our models, the while cycle also runs in the proposed $O(|V| + |E|)$ time. ∎

Claim 10. *The optimum of 2D1R can be* 3*-approximated in linear time:* $O\big(|V| + |E|\big)$.

Proof: We create levels $\mathcal{L}(i)$ as in Alg. 1. For a level i we partition tables into groups $V_{w,k}$ according to width as described in §IV-C. First nodes in $V_{w,0}$ are embedded in $\lceil \sum_{v \in V_{w,0}} n_v / n_S \rceil$ stages, applying $hsplit$ if needed. Next, nodes in $V_{w,1}$ are embedded. They need $\lceil \sum_{v \in V_{w,1}} n_v/2 \rceil$ rows, starting with the first row unoccupied by $V_{w,0}$. Note that the last row may not be at least half full, because the second column can remain empty. Generally, when we pack MATs in $V_{w,k}$ assume that d columns of length $1/2^{k-1}$ are left empty in the last row after embedding nodes in $V_{w,k-1}$. Then if $\sum_{v \in V_{w,k}} n_v < 2d$, then we cut these tables into 1-row pieces and embed them all into this last row. Otherwise, we open $\lceil (\sum_{v \in V_{w,k}} n_v - 2d)/2^k \rceil$ new rows and embed the tables continuously (again applying $hsplit$ if needed) into 2^k columns nearly equally, i.e., such that only columns in the last row may remain empty.

To show 3-approximation, let OPT denote the minimal number of stages the TDG can be mapped to. Let h denote the number of levels in the TDG, and for each level $\mathcal{L}(i)$, let $\mathcal{S}[\mathcal{E}_i]$ denote the last stage where nodes of the level are matched. Since at least half of the memory of each row of each stage $\mathcal{S}[\mathcal{E}_{i-1}+1], \dots, \mathcal{S}[\mathcal{E}_i]$ is used, $\mathcal{E}_h - h \leq 2 \cdot \text{OPT}$. Also, $\text{OPT} \geq h$, since we need at least as many stages as the number of levels we have, so $\mathcal{E}_h \leq 3 \cdot \text{OPT}$. To decipher the *complexity* of the 2D1R version of Alg. 2, we only analyze the additional calculations which have to be done compared to the 1D1R-$hsplit$ version Alg. 2 (that runs in $O\big(|V|+|E|\big)$ by Thm. 9). Partitioning of tables in each level by their width can be done in $O(|V|)$ total time, since we supposed that w_S/w_v and w_T/w_v are $O(1)$. For each table, we need $O(1)$ time to compute its start and end coordinates, meaning $O(|V|)$ complexity. ∎

Claim 11. *The optimum of 2D2R can be* 8*-approximated in linear time:* $O\big(|V| + |E|\big)$. *If* $w_T \geq w_S$*, it can be* 5*-approximated in the same complexity.*

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

First, we concentrate on a given level $\mathcal{L}(i)$. Let us recall that the last stage occupied by stage $\mathcal{L}(i)$ is denoted by $\mathcal{S}[\mathcal{E}_{i-1}]$. When embedding to TCAMs (just like as in 2D1R version of Alg. 2), we scale the width unit to the width of the TCAM.

We classify the set of MATs in the current level into three groups S, T, and A, according to the type of memory they can be embedded into (S for SRAM, T for TCAM and A for all), and we also partition each set further into groups S_k, T_j and $A_{k,j}$ according to widths as follows.

Let S_k denote the set of MATs in S of length $w_S \cdot [1/2^{k+1}, 1/2^k)$ and let T_j denote the set of MATs in T of length $w_T \cdot [1/2^{j+1}, 1/2^j)$. Finally, we partition the MATs in A according to both their relative widths to w_S and w_T: $A_{k,j}$ denotes the set of MATs in A of length $w_S \cdot [1/2^{k+1}, 1/2^k) \cap w_T \cdot [1/2^{j+1}, 1/2^j)$; we note that for a fixed value k there are at most two possible values j such that $A_{k,j}$ is not empty.

We state the following. 1) With packing T_j in 2^j columns (for $j = 0, 1, \dots$, similarly to the 2D1R case), T, for some t_i, occupies stages $\mathcal{S}[\mathcal{E}_{i-1}+1], \dots, \mathcal{S}[\mathcal{E}_{i-1}+t_i]$, that (apart from the last of them) have their TCAMs at least half full. 2) If the width of SRAMs is greater than the width of TCAMs (i.e., $w_S > w_T$), then S may not be empty. With packing S_k in 2^k columns (for $k = 0, 1, \dots$), S, for some s_i, occupies stages $\mathcal{S}[\mathcal{E}_{i-1}+1], \dots, \mathcal{S}[\mathcal{E}_{i-1}+s_i]$, that (apart from the last

of them) have their SRAMs at least half full. If S is empty, let $s_i = 1$. 3) Starting from stage $\mathcal{S}[\mathcal{E}_{i-1} + \max\{t_i, s_i\} + 1]$, we pack the MATs $A_{k,j}$ (for $k = 0, 1, \ldots$ with appropriate j and $j+1$ values) first in 2^k columns in SRAMs in a stage then in 2^j columns in TCAMs in the same stage. Thus A, for some l_i, occupies stages $\mathcal{S}[\mathcal{E}_{i-1} + \max\{t_i, s_i\} + 1], \ldots, \mathcal{S}[\mathcal{E}_{i-1} + \max\{t_i, s_i\} + l_i]$, that (apart from the last of them) have both their SRAMs and TCAMs at least half full.

We can see by Claim 10 that $t = \sum_{\text{levels}}(t_i - 1) \leq 2 \cdot \text{OPT}$ (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, $\sum_{\text{levels}}(s_i - 1) \leq 2 \cdot \text{OPT}$. This means $\sum_{\text{levels}}(\max\{t_i, s_i\} - 1) \leq 4 \cdot \text{OPT}$.

Similarly, tables that can be mapped to both TCAMs and SRAMs, fill at most another $\leq 2 \cdot \text{OPT}$ stages at least half full (i.e. $\sum_{\text{levels}}(l_i - 1) \leq 2 \cdot \text{OPT}$), meaning at most $6 \cdot \text{OPT}$ stages. For each level, we have not counted i) the last stage it uses, and (maybe) ii) the last stage storing SRAM-only or TCAM-only tables (i.e., its $\max\{t_i, s_i\}^{\text{th}}$ stage). Since there are $\leq$ OPT levels, we can conclude that the algorithm uses at most $(6 + 2) \cdot \text{OPT} = 8 \cdot \text{OPT}$ stages. In case of $w_T \geq w_S$, $s_i \equiv 0$ for all i, and our upper bound shrinks to $5 \cdot \text{OPT}$, because then 1) (implicitly) SRAM-only tables do not exist, and thus, they do not have to be taken into account, and 2) if neither the level's $t_i + l_i{}^{\text{th}}$ stage nor the TCAMs of its $t_i{}^{\text{th}}$ stage are at least half full, then by re-allocating some entries from the $t_i + l_i{}^{\text{th}}$ stage to the $t_i{}^{\text{th}}$ stage, we may either empty the $t_i + l_i{}^{\text{th}}$ stage, or can make $t_i{}^{\text{th}}$ stage at least half full. Note that in MATs involved in this entry-reallocations are in the end cut in $\leq 4 \cdot w_S/w_v = O(1)$ pieces, and we only need to specify the start and end places of the reallocated entries.

Runtime complexity of the 2D2R version of Alg. 2: for each level, sets T_k and S^j can be determined in linear time (similarly to sets $V_{w,k}$ in 2D1R). A can be partitioned to sets A_k of MATS having widths $w_T \cdot [1/2^{k+1}, 1/2^k)$ in this time. Then, it is easy to find the right j s.t.: $w_S/2^j \in w_T \cdot [1/2^{k+1}, 1/2^k)$, with which we can subdivide each set A_k into two (possibly empty) subsets regarding their width relative to w_T. After computing the levels, we assign each table to a partition ≤ 2 times. Thus, the 2D2R version of Alg. 2 runs in $O(|V|+|E|)$. ∎

Claim 12. *The optimum of 2D2R-T/S can be* 9*-approximated in linear time:* $O(|V| + |E|)$. *If* $w_T \geq w_S$*, it can be* 6*-approximated in the same complexity.*

Proof: We again create levels $\mathcal{L}(i)$ as in Alg. 1. Similarly to the previous algorithm for 2D2R, in each level i, we classify the set of MATs into three groups T^i, S^i and A^i, and we also partition each set further into groups T^i_j, S^i_k and $A^i_{k,j}$ according to widths as in the previous algorithm. First we embed MATs in T^i into TCAMs in stages $\mathcal{S}(\mathcal{E}_{i-1}+1), \ldots, \mathcal{S}(\mathcal{E}_{i-1}+t_i)$ similarly to the previous algorithm, the only difference is that when the number of tables embedded in a stage reaches τ, we open a new stage. Then we embed MATs in S^i into SRAMs of stages $\mathcal{S}(\mathcal{E}_{i-1} + t_i + 1), \ldots, \mathcal{S}(\mathcal{E}_{i-1} + t_i + s_i)$ the same way. Finally, MATs in A^i are embedded in stages $\mathcal{S}(\mathcal{E}_{i-1} + t_i + s_i + 1), \ldots, \mathcal{S}(\mathcal{E}_{i-1} + t_i + s_i + a_i)$. Now prove the approximation ratio. Let $\mathcal{S}^i_{T,w}$ denote the set of stages where TCAMs are at least half full with MATs from T_i with respect to width, and let $\mathcal{S}^i_{T,\tau}$ denote the set of stages *not* in $\mathcal{S}^i_{T,w}$ that contain τ MATs from T_i. Note that stages $\mathcal{S}(\mathcal{E}_{i-1} + 1), \ldots, \mathcal{S}(\mathcal{E}_{i-1} + t_i - 1)$ belong to exactly one of these sets. Then $\sum_{\text{levels}}(|\mathcal{S}^i_{T,w}|) \leq 2 \cdot \text{OPT}$. We can similarly define sets $\mathcal{S}^i_{S,w}$, $\mathcal{S}^i_{S,\tau}$ and $\mathcal{S}^i_{A,w}$, $\mathcal{S}^i_{A,\tau}$ for stages embedding S^i and A^i, respectively. We also have $\sum_{\text{levels}}(|\mathcal{S}^i_{S,w}|) \leq 2 \cdot \text{OPT}$ and $\sum_{\text{levels}}(|\mathcal{S}^i_{A,w}|) \leq 2 \cdot \text{OPT}$. There are at most three stages on every level that do not belong to any of the set above: $\mathcal{S}(\mathcal{E}_{i-1} + t_i), \mathcal{S}(\mathcal{E}_{i-1} + t_i + s_i)$ and $\mathcal{S}(\mathcal{E}_{i-1} + t_i + s_i + a_i)$. Stages $\mathcal{S}(\mathcal{E}_{i-1} + t_i)$ and $\mathcal{S}(\mathcal{E}_{i-1} + t_i + s_i)$ can be unified into one stage if the number of tables is at most τ. Otherwise they can be unified into two stages such that one of them contains τ stages, thus it can be added to $\mathcal{S}^i_{A,\tau}$ and eventually there are at most two stages on every level that do not contain τ tables and are not at least half full. The total number of the latter is at most $2 \cdot \text{OPT}$. Finally, the number of stages containing τ tables can be estimated together: since $hsplit$ is only applied when a stage becomes at least half full, every original MAT has a piece in at most one stage in $\mathcal{S}^i_{T,\tau}$, thus $\sum_{\text{levels}}(|\mathcal{S}^i_{T,\tau}| + |\mathcal{S}^i_{S,\tau}| + |\mathcal{S}^i_{A,\tau}|) \leq \text{OPT}$. All in all, the total number of stages is at most $(2+2+2+2+1) \cdot \text{OPT} = 9 \cdot \text{OPT}$. As for the case when $w_T \geq w_S$ that is, $S^i = \emptyset$, we can create sets $\mathcal{S}^i_{T,w}, \mathcal{S}^i_{T,\tau}$, $\mathcal{S}^i_{A,w}, \mathcal{S}^i_{A,\tau}$ as in the first case and embed them the same way. Stages $\mathcal{S}(\mathcal{E}_{i-1} + t_i)$ and $\mathcal{S}(\mathcal{E}_{i-1} + t_i + a_i)$ may not be part of these sets. Similarly to the previous model we can rearrange these stages in order to have at most one such stage. If the TCAM part of $\mathcal{S}(\mathcal{E}_{i-1} + t_i + a_i)$ is empty, we can unify it with $\mathcal{S}(\mathcal{E}_{i-1} + t_i)$. If it is not empty, then its SRAM part is at least half full. We can re-embed these stages into stage $\mathcal{S}(\mathcal{E}_{i-1} + t_i)$ the following way: we copy the SRAM part of stage $\mathcal{S}(\mathcal{E}_{i-1} + t_i + a_i)$ and consider the union of MATs in TCAMs $\mathcal{S}(\mathcal{E}_{i-1} + t_i)$ and $\mathcal{S}(\mathcal{E}_{i-1} + t_i + a_i)$. We start embedding them into the TCAM part of stage $\mathcal{S}(\mathcal{E}_{i-1} + t_i)$ the same way as before until 1) all tables are embedded 2) the TCAM (and thus the whole stage) becomes half full, and we can add it to $\mathcal{S}^i_{A,w}$ 3) the total number of tables reaches τ and we can add it to $\mathcal{S}^i_{A,\tau}$. In the latter two cases we embed the remaining tables to stage $\mathcal{S}(\mathcal{E}_{i-1} + t_i + a_i)$. We have similarly that $\sum_{\text{levels}}(|\mathcal{S}^i_{T,w}|) \leq 2 \cdot \text{OPT}$ and $\sum_{\text{levels}}(|\mathcal{S}^i_{A,w}|) \leq 2 \cdot \text{OPT}$ and $\sum_{\text{levels}}(|\mathcal{S}^i_{T,\tau}| + |\mathcal{S}^i_{A,\tau}|) \leq \text{OPT}$. Moreover, for every level there is at most one stage not considered above, giving $(2 + 2 + 1 + 1) \cdot \text{OPT} = 6 \cdot \text{OPT}$. ∎

Now, we turn to the **dRMT models**. First, we describe a simple approximation algorithm to model WIDTH, which is based on an observation that the precedence constraints can be made irrelevant in terms of minimizing period P.

Theorem 13. *For model WIDTH, a scheduling having a period at most* $3/2$ *times the optimal can be calculated in* $O(|V| + |E|)$ *time.*

Proof: First, we ignore the precedence constraints, and consider two bin-backing problems with node widths as object weights and $\overline{A}, \overline{M}$ as bin capacities. Using the linear-time $3/2$-approximation algorithm of [26], we separately sort the match and action nodes in a number of M' and A' bins, respectively. Thus, a scheduling with period $P = \max\{M', A'\}$ would be a $3/2$-approximation on the optimal period. Now we show

that such a scheduling exists. Let B_m^i and B_a^i denote the i^{th} bin of match and action nodes, respectively. For each $i \in \{1,\dots,P\}$, we assign B_m^i and B_a^i to residue class i. Now, we take into count the precedence constraints again, and take an arbitrary topological order of the nodes $v_1,\dots,v_n$. For each $j \in \{1,\dots,n\}$, we schedule v_j (being part of a batch B_v^i) to the smallest positive integer clock cycle $\mathsf{S}(v_j)$ that is $\equiv i \mod P$, and is greater or equal with $\mathsf{S}(v)+l(v)$, for each node v directly preceding v_j in the ODG (that is, $(v,v_j) \in E$). Note that if v_j does not have any in-arc, it is scheduled to i. ■

In the following, we describe approximation algorithms for models IPC1, W-IPC1, and W-IPC2. The idea is to find a proper order of the nodes (called simplified scheduling) that can be transformed into a scheduling in a straightforward way.

Definition 3. *Function $\sigma : V \to \mathbb{N}^+$ is a **simplified scheduling**, if*

1) $\sigma(m) \neq \sigma(a)$ *for every* $m \in V_M, a \in V_A$,
2) $\sigma(v_j) - \sigma(v_i) \geq 1$ *for every arc* $(v_i,v_j) \in E$,
3) *if* $\sigma^{-1}(k) = \emptyset$ *for a* $k \in \mathbb{N}^+$, *then* $\sigma(v) < k$ *for* $\forall v \in V$,
4) $\forall t \in \mathbb{N}^+ : \sum_{m \in V_M,\, \sigma(m)=t} w_m \leq \overline{M}$,
5) $\forall t \in \mathbb{N}^+ : \sum_{a \in V_A,\, \sigma(a)=t} w_a \leq \overline{A}$.

*Let $L(\sigma)$ denote the **length** of the simplified scheduling, so the largest clock cycle that has an embedded node: $L(\sigma) = \max\{i | \sigma^{-1}(i) \neq \emptyset\}$.*

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

Lemma 14. *For dRMT models IPC1 and W-IPC1, a given simplified scheduling σ can be transformed to a scheduling with a period of $P := \max(A, M)$ in $O(|E|+|V|)$ time.*

Proof: We *expand* the simplified scheduling σ to a scheduling with period P in linear time. Formally, a scheduling S is an *expansion of a simplified scheduling* σ if there exists a strictly monotone function $f : \mathbb{N}^+ \to \mathbb{N}^+$ such that $\mathsf{S}(v) = f(\sigma(v))$. We have to show that σ has a feasible P-periodic expansion. For this, 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. We determine values $f(1),\dots,f(L(\sigma))$ in this order, and we fill up modulo classes with match and action nodes in the order of $1,2,\dots,P$ (we will use each modulo class once for both match and action nodes, respectively). Let the smallest modulo classes not used by match and action nodes be p_m and p_a, respectively. Let $f(1) = 1$, and for $1 < i \leq L(\sigma)$ we do as follows. If there is an arc entering a node in $\sigma^{-1}(i)$ then the earliest clock cycle $\sigma(i)$ may be embedded in is $\delta(i) := \max\{f(\sigma(v)) + l(v) | vw \in E, \sigma(w) = i\}$, otherwise $\delta(i) := f(i-1)+1$. Supposing $\sigma^{-1}(i)$ is a set of match nodes, we assign these nodes to clock cycle $f(i) := \min\{k \geq \delta(i) |\ k \equiv p_m\}$, and increase p_m by 1. Similarly, if $\sigma^{-1}(i)$ is a set of action nodes, $f(i) := \min\{k \geq \delta(i) | k \equiv p_a\}$, and p_a is increased by 1. Note that there is always some modulo class left to embed the sets of nodes into, since $P \geq M$ and $P \geq A$, and former clock cycles of the same type cannot cover all residue classes modulo P. Also note that, in every congruence class, there is at most 1 CPU cycle where match and action nodes are embedded, thus the expansion is valid also if IPC constraints are present. Finally, the width constraints were already tackled by the simplified scheduling, thus the resulting scheduling respects width constraints also. The method is summarized in Alg. 4. Determining values $\delta(i)$ and $f(i)$ take $O(|E|+|V|)$. ■

Theorem 15. *For model W-IPC1, a scheduling having a period at most 4 times the optimal can be calculated in $O(|V|+|E|)$.*

Proof: Inspired by Lemma 14, in Alg. 3, we first find a simplified scheduling σ. We create levels $\mathcal{L}(i)$ as in Alg. 1. For each level we embed match and action nodes in separate clock cycles, applying the linear-time algorithm in [26]. Thus, for every level, there is at most one clock cycle containing match nodes that is not at least half full (and the same applies for action nodes). Now we prove that the algorithm is a 4-approximation. We partition the clock cycles into four groups. Let H_m and H_a denote those clock cycles that are at least half full with match and action nodes, respectively, and similarly, let N_m and N_a denote the list of those that are not half full. We can assume w.l.o.g. that $M \geq A$. From Lemma 14, we get that the constructed simplified scheduling can be expanded into feasible scheduling with period M. Let P_o denote the optimal period for the problem. We know that $P_o \geq \sum_{m \in V_M} w_m / \overline{M}$ (note that the under-estimation of the optimal period P_o was indifferent to any IPC constraint). Since $\sum_{v \in V_M} w_m/\overline{M} \geq \sum_{m \in \sigma^{-1}(H_m)} w_m/\overline{M} \geq |H_m|/2$ and we get $|H_m| \leq 2P_o$. Now we give an upper bound for $|N_m|$. If the number of levels is denoted by h, then there is a path P of length h in D_O. 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 $|N_m| \leq |V(\mathcal{P})| \leq 2P_o$ and so $|M| = |H_m| + |N_m| \leq 4P_o$, which proves the theorem. The runtime of the algorithm is $O(|V|+|E|)$. ■

Theorem 16. *Model IPC1 can be $3-$approximated in $O(|V|+|E|)$ time.*

Sketch of proof: We can simplify the previous algorithm by noting that all nodes have unit width, so we can embed nodes to get full clock cycles (with either match or action nodes only), apart from the last clock cycle in each level and each type. Let F_m and F_a denote the set of full clock cycles. Then we have $|F_m| \leq P_o$, which gives a 3-approximation. ■

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

Theorem 17. *Model W-IPC2 can be $8-$approximated in $O(|V|+|E|)$ time.*

Proof: Let P_1^{opt} and P_2^{opt} denote the optimal periods for W-IPC1 and W-IPC2, respectively for models with the same input parameters (except IPC). Let S denote a feasible P-periodic scheduling for W-IPC2. We transform S into an S' feasible $2P$-periodic scheduling for W-IPC1. Since IPC $= 2$, for each residue class i modulo P, there are at most two non-empty clock cycles c_1, c_2 congruent i modulo P. If $\mathsf{S}(v) = c_1$ then we define $\mathsf{S}'(v) := 2c_1$ whereas if $\mathsf{S}(v) = c_2$ then $\mathsf{S}'(v) := 2c_2 - 1$, so $P_1^{\text{opt}} \leq 2P_2^{\text{opt}}$. Let P^* denote the period given in Thm. 15. Then $P^* \leq 4P_1^{\text{opt}} \leq 8P_2^{\text{opt}}$. ■