P4CGO: Control Plane Guided P4 Program Optimization

Chenan Wen
Purdue University
West Lafayette, Indiana, USA
wen163@purdue.edu

Zhuocong Li
Purdue University
West Lafayette, Indiana, USA
li3975@purdue.edu

Syed Usman Jafri
Purdue University
West Lafayette, Indiana, USA
jafri3@purdue.edu

Xiaokang Qiu
Purdue University
West Lafayette, Indiana, USA
xkqiu@purdue.edu

Sanjay Rao
Purdue University
West Lafayette, Indiana, USA
sanjay@purdue.edu

Abstract

Software-defined networking (SDN) in conjunction with programmable switches revolutionizes network management, yet crafting optimal switch configurations remains complex. Traditional P4 optimizations rely on data plane level tuning. In this paper, we argue an essential piece for such optimizations is the control plane itself. We present P4CGO, a P4 compilation framework which focuses on realizing specifications based on control policies. P4CGO leverages user-defined objective functions and control plane policies to guide P4 program optimization through table merging and splitting. We have prototyped P4CGO and applied it solving real-world policy optimization problems.

CCS Concepts

• **Mathematics of computing** → **Network optimization**; • **Networks** → **Programmable networks**; *Programming interfaces*; • **Software and its engineering** → Domain specific languages.

Keywords

Programmable Switch, Control Plane, Optimization, Formal Methods

ACM Reference Format:
Chenan Wen, Zhuocong Li, Syed Usman Jafri, Xiaokang Qiu, and Sanjay Rao. 2024. P4CGO: Control Plane Guided P4 Program Optimization. In *SIGCOMM Workshop on Formal Methods Aided Network Operation (FMANO '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3672199.3673892

1 Introduction

The wide-spread adoption of Software-Defined Networking (SDN) has revolutionized the way networks are managed and operated. Originating from OpenFlow [11], SDN separates the control plane from the data plane to allow for centralized control, simplified network management and efficient policy deployment. P4 [2] as a domain-specific language for programming packet processing pipelines, complements these capabilities by offering fine-tuning data plane managements.

With the help of P4, Programmers can implement complex packet handling logic by setting policy tables, match-keys, actions and table sizes. However, as a hardware-specific language, programmers need to manually optimize resource usage and ensure compatibility with the underlying hardware resources. Researchers are actively exploring techniques to enhance P4 programs' packet processing efficiency while guaranteeing the resource capabilities. However, most optimizations focus solely on the program level without considering the control policy content. We argue that a crucial piece for such optimization is the control plane itself. Given typical control planes, we may wish to make very different decisions about the number and sequence of tables. In studies altering table contents [9, 20], a lack of vision in the control plane often leads to worst-case assumptions.

To bridge the gap between specification and optimal realization, we present P4CGO that takes both control plane and data plane as input. It leverages control policies and uses formal approaches to guide the automatic optimization by table merging and table splitting. This process generates optimal equivalent data plane tables along with corresponding optimized control plane policies, ensuring efficient resource utilization and adherence to specified network policies.

2 Motivation

In this section, we motivate our work through two examples.

2.1 Why current work falls short?

Pipeleon [20] bridges the gap between P4 programs and their performance on SmartNICs by contributing an automated SmartNIC optimization framework with profile-guided, performance-oriented P4 optimizations. It uses table reordering, table caching and table merging methods to optimize the P4 program on SmartNICs. It enables merging two tables into one TCAM table using cross product to shorten the table lookup pipeline. Since they do such optimization at data plane level, agnostic to the control plane policies, they generate the merged TCAM with worst case size.

Similar to Pipeleon's table merging, Cetus [9] also tries to merge tables. Since Cetus focuses on optimizing programs on switches instead of SmartNICs, it aims to to shorten the critical path on switch pipeline by removing dependencies. Since the dependencies could be caused by WAW, WAR, RAW in key match and RAW in action,

FMANO '24, August 4–8, 2024, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0714-8/24/08
https://doi.org/10.1145/3672199.3673892

Table 1: Tables in An Example P4 Program

(a) ACL Table

Protocol	Action
ip	allow
tcp	allow
udp	allow
*	drop

(b) QoS Table

SrcIP	DstIP	Action
192.168.1.2	10.0.0.1	forward
131.5.23.*	10.0.0.1	forward
131.196.*.*	10.0.0.1	forward
*	*	drop

(c) NAT Table

SrcIP	SrcPort	Action
192.168.1.2	8080	translate
192.168.1.3	8080	translate
172.16.*.*	8080	translate
*	*	drop

Table 2: Merged Table

Protocol	SrcIP	SrcPort	DstIP	Action
ip	192.168.1.2	8080	10.0.0.1	forward&translate
tcp	192.168.1.2	8080	10.0.0.1	forward&translate
udp	192.168.1.2	8080	10.0.0.1	forward&translate
*	*	*	*	drop

Cetus implements a merging agenda to modify the match-keys and action assignments. Considering the limited stage memory size and PHV capacity, Cetus encodes hardware resource size constraints by building a binary decision tree for optimization strategy and prunes the branches that violate basic memory and stage constraints with the help of SMT solvers [4]. It also provides a control plane API for tables to be deployed.

Although merging tables for P4 programs can be beneficial for table latency since it reduces the memory lookups and required stages, improving cache efficiency, it also introduces memory overhead by applying entry multiplications. In certain cases, the act of merging tables may necessitate a shift in memory types from SRAM to TCAM. When evaluating the trade-offs, the table's policy content needs to be taken into consideration. However, Pipeleon and Cetus calculate only the worst-case memory usage due to their obliviousness of control planes.

2.2 Why control-plane guided optimization?

As shown in Table 1, a P4 program comprising three tables is undergoing optimization. Our goal is to merge tables to use less stages. Assuming there are no dependencies among these actions, if we are unaware of the control plane policy, the size of the merged table is determined by multiplying the sizes of all the tables being merged. Any two of these table will be merged into a 9-entry table (excluding the default entry). All three tables will be merged into a 27-entry table. However, if we have the vision in control plane, we will find out that only certain flow can be forwarded and translated. If we merge Tables 1a, 1b, and 1c all together, we can obtain Table 2 with only 3 entries. While Pipeleon also noted that all merged tables would be TCAM tables due to the introduction of wildcards in the merging process, in this particular example, we can utilize SRAM to store the merged table instead.

When previous works attempted to merge tables, if the table size exceeds the memory limit, they will discard the merging option by pruning the decision tree [9, 20], hence losing the opportunity of global optimization for the entire program. Now consider a data plane policy represented as an ACL table shown in Table 3. This table is to permit traffic from four specific source IP addresses, each with permission for four source ports: 10, 20, 30 and 40. Therefore, the comprises 16 entries (excluding the default entry) which may exceed the TCAM memory limit. By observing the table content, it is

Table 3: ACL Table

SrcIP	SrcPort	Action
192.168.1.0	10	allow
192.168.1.0	20	allow
192.168.1.0	30	allow
192.168.1.0	40	allow
131.5.23.*	10	allow
131.5.23.*	...	allow
198.51.100.1	...	allow
172.31.255.2	...	allow
*	*	drop

Table 4: Splitted Tables

(a) Sub-Table 1

SrcIP	Action
192.168.1.0	Flag=F0
131.5.23.*	Flag=F0
198.51.100.1	Flag=F0
172.31.255.2	Flag=F0
*	Flag=F1

(b) Sub-Table 2

Flag	SrcPort	Action
F0	10	allow
F0	20	allow
F0	30	allow
F0	40	allow
*	*	drop

notable that the multiplication between SrcIP and SrcPort causes the memory blowup. However, one may decouple the table by splitting the ACL table into two sub-Tables 4a and 4b. The two sub-tables consume approximately 34% of the original table's memory.

The goal of this paper is to automate this process and discover optimal table merging and splitting algorithmically.

3 System Design

In this section, we present the design of P4CGO, a control plane guided optimization framework. It takes multi-table control policies corresponding to a P4 program as input and first merges them into a single big policy table. Then it tries to split the merged policy table to accommodate the target hardware capacity through a process based on automaton minimization. By the merging and splitting steps P4CGO transforms the input control plane into a *provably optimal* implementation for target hardware based on a given cost function. Figure 1 shows the entire workflow that consists of following phases:

- First, generate a merged policy table from input control plane policies. The merging process involves combining multiple tables through multiplication and checking reachability for each branch.
- Second, expand the generated policy table from first step into a non-overlapped table and further transform into a regular-expression representation.
- Third, build a deterministic automaton to accept the RegEx and perform automaton minimization.
- Last, enumerate all possible ways to split the automaton into sub-automata by cutting from the states. Each split yields a candidate flow of tables through SMT solving. P4CGO returns the one with the minimal cost as the optimal solution.

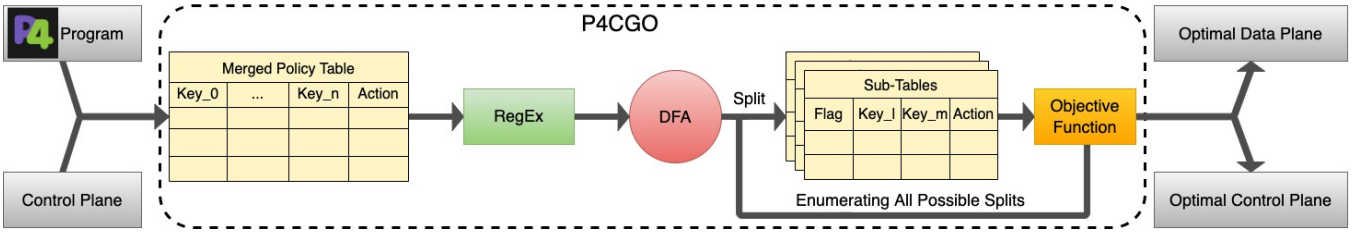


Figure 1: P4CGO Workflow

3.1 Transforming Policy into RegEx Representation

Users can provide the control plane as an input to P4CGO. The provided policy table usually contains multiple rules. Each rule contains multiple key fields and a corresponding action. After receiving a policy table, P4CGO will use a set of symbols to encode these rules' *non-overlapped* part and transform the ruleset into a regular expression. When a key has multiple values, the compiler will find all distinct areas in *Venn Diagram* and encode each area into a symbol $SrcIP_i$. For example, if there are two overlapped value A and B in key $SrcIP$, P4CGO encodes all distinct areas $(A \wedge B)$, $(A \wedge \neg B)$, $(\neg A \wedge B)$, $(\neg A \wedge \neg B)$ into $SrcIP_0$, $SrcIP_1$, $SrcIP_2$, $SrcIP_3$, respectively. We call each distinct area *virtual value* since it represents a set of distinct ternary entries and sometimes can't be represented by one ternary value, e.g., $(0***\wedge\neg 0000)$. After encoding, each ternary value in the table is encoded into a set of symbols. In previous example, $A = \{SrcIP_0, SrcIP_1\}$, $B = \{SrcIP_0, SrcIP_2\}$. If we use $*$ to represent all $SrcIP$ then $* = \{SrcIP_0, SrcIP_1, SrcIP_2, SrcIP_3\}$.

After we encode each distinct combination, each rule can be expressed by the Cartesian product of all its symbol sets. For a rule $E_n : SrcIP = A, \{SrcPort = P, Action = D\}$, if we have encoding $A = \{SrcIP_0, SrcIP_1\}$, $P = \{SrcPort_0\}$, $D = \{Act_0\}$, then we can generate RegEx

$$E_n = (SrcIP_0\ SrcPort_0\ Act_0) \mid (SrcIP_1\ SrcPort_0\ Act_0) \quad (3.1)$$

to represent E_n.

If we apply same RegEx generation for every rule in the table, we can transform the original policy table into a RegEx. However, rules in policy tables are ordered in priority. If two rules E_m and E_n are overlapped and E_m occurs earlier in the table, for the overlapped part, the action is decided by rule with higher priority, i.e., E_m. Since we want the generated regex to be equivalent, we need to respect the original priority and remove the overlapped part from E_n. For example, for previous generated rule (Eq 3.1), if there is another higher priority rule $E_m = (SrcIP_0\ SrcPort_0\ Act_1)$, we need to remove the overlapped key part and make $E_n = (SrcIP_1\ SrcPort_0\ Act_0)$.

3.2 Splitting DFA

After we transform the policy table into equivalent regex, we can further transform it to an automaton representation. Given the RegEx, we can build a deterministic finite automaton (DFA) to accept the policy. Through standard DFA minimization, we can obtain an automaton with minimal number of states. Figure 2 is such a minimized DFA example. This merged policy table has four keys: $SrcIP$, $SrcPort$, $DstIP$, $DstPort$ and one $Action$. If we use

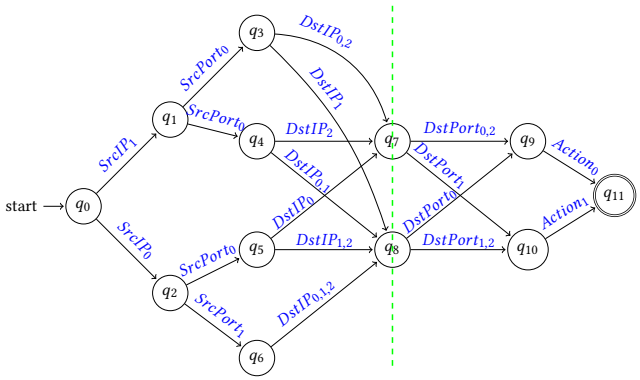


Figure 2: DFA Split

RegEx 11000 to represent

$$E_0 = (SrcIP_1\ SrcPort_1\ DstIP_0\ DstPort_0\ Action_0)$$

for simplicity, from top to bottom the RegEx this DFA accepts can be written as

$$11000 \mid 11020 \mid \ldots \mid 01221$$

The number of terms of this representation is the total number of paths from the initial state to a final state.

The automaton effectively represents a non-overlapping, virtual-value policy table with each entry representing an accepting path of the automaton. As mentioned in §2.2, the merged policy table size can exceed the hardware capacity. To address this issue, P4CGO splits the DFA between key fields and generates multiple sub-tables. As shown in Figure 2, if we cut vertically between $DstIP$ and $DstPort$, the DFA will be split into two sub-DFAs. We further convert the sub-DFAs to *virtual value* tables as shown in Tables 5a and 5b, respectively. The generated sub-tables follow a sequential order, as each table uses the flag from the previous table as a key, representing the end state of the previous sub-DFA. If there are n key fields in the policy, the number of possible cuts is 2^{n-1}. In our implementation we enumerate all possible cuts of up to n sub-DFAs where n is the stage number limited by hardware.

3.3 Generating Control Plane Tables

The sub-tables generated from DFA splitting may contain non-overlapping virtual entries which are not necessarily legal ternary or exact entries. So the next step is to convert them back to standard overlapping policy tables with priorities. As mentioned in §3.1, each key's *virtual values* can be generated by finding all value

Table 5: Tables Generated from Split DFAs

(a) Sub-Table 1

SrcIP	SrcPort	DstIP	Action
$SrcIP_1$	$SrcPort_1$	$DstIP_0$	Flag=q_7
$SrcIP_1$	$SrcPort_1$	$DstIP_2$	Flag=q_7
...	...	...	...
$SrcIP_0$	$SrcPort_1$	$DstIP_2$	Flag=q_8

(b) Sub-Table 2

Flag	SrcIP	Action
q_7	$DstPort_0$	$Action_0$
q_7	$DstPort_2$	$Action_0$
...	...	...
q_8	$DstPort_2$	$Action_1$

combinations. In the worst case in which all value combinations are satisfiable, if there are m different values in a key, the number of non-overlapped virtual values for the same key can be 2^m, due to the Cartesian product of all keys' virtual value sets. Consequently, a naïve conversion can experience an exponential blowup in the number of entries, which leads to unsatisfactory tables as output.

To address this problem, P4CGO takes the following steps to convert the raw, non-overlapped table to a minimal, overlapped tables. First, it groups non-overlapping entries by flag keys. Then for each group, it uses a reduction algorithm to merge the *virtual* entries. The reduction algorithm is a special case of the *Quine McCluskey Method (QMC)* [10, 14].

Table 6 shows an example raw table that covers all combinations of ternary matches A, B and C. The table shows only the portion that starts from flag q_7. Note that the negation of a ternary value (e.g., $\neg A$) cannot be directly matched using a single entry. To this end, we depict all combinations involving negative values using a conjunction of literals such that all but the last one is a positive one. Intuitively, the conjunction will be matched through a sequence of entries, each negative literal for an unmatched entry and the last, positive literal for the matched entry. Below are several examples:

$$(\neg ABC) \equiv \neg(ABC) \wedge (BC) \tag{3.2}$$

$$(\neg A \neg BC) \equiv \neg(AC) \wedge \neg(BC) \wedge (C) \tag{3.3}$$

$$(\neg A \neg B \neg C) \equiv \neg(A) \wedge \neg(B) \wedge \neg(C) \wedge \top \tag{3.4}$$

Based on the representation above, we construct a layered decision diagram as depicted in Figure 3. In this diagram, each node represents a condition combination and labeled with the corresponding action. Note that lower level nodes have more values to match (e.g., node ABC at the bottom has to match all three ternary values) and have higher priority than upper level nodes. In other words, every node excludes the values of all its descendants (e.g., node AB represents $(AB) - (ABC)$, or $(AB\neg C)$). Consequently, every node is equivalent to a combination shown in Table 6, and can be labeled with the corresponding action from Table 6. The crux of the compression algorithm is based on the fact that if a node shares the same action as its parent, it can be integrated into the parent node by removing one negation term from the parent's expression. Consequently, we can perform a reduction on the diagram. For example, as nodes ABC, AC, and AB are all labeled with action q_0, they can be merged into their ancestor node A. Similarly, Node BC can be merged into node B as they both trigger action q_1. The remaining four nodes still effectively cover all condition combinations:

$$\text{Node } \top \equiv \neg(A) \wedge \neg(B) \wedge \neg(C) \wedge \top \tag{3.5}$$

$$\text{Node } A \equiv (A) \tag{3.6}$$

$$\text{Node } B \equiv \neg(AB) \wedge (B) \tag{3.7}$$

Table 6: Non-Overlapped Table before Compression (only entries starting with flag q_7 shown)

Flag	Key	Action
...	...	...
q_7	ABC	Flag=q_0
q_7	$\neg ABC$	Flag=q_1
q_7	$A\neg BC$	Flag=q_0
q_7	$AB\neg C$	Flag=q_0
q_7	$\neg A\neg BC$	Flag=q_1
q_7	$\neg AB\neg C$	Flag=q_1
q_7	$A\neg B\neg C$	Flag=q_0
q_7	$\neg A\neg B\neg C$	Flag=q_2
...	...	...

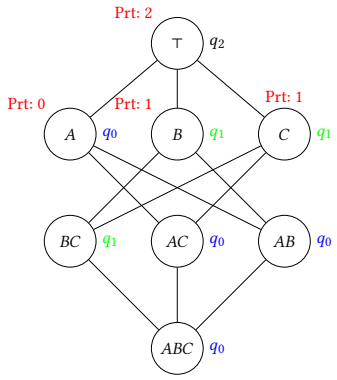


Figure 3: Reduction Diagram

Table 7: Overlapped Table after Compression (cf. Table 6)

Node	Priority	Flag	Key	Action
A	0	q_7	A	Flag = q_0
B	1	q_7	B	Flag = q_1
C	1	q_7	C	Flag = q_1
$\top$	2	q_7	$*$	Flag = q_2

$$\text{Node } C \equiv \neg(AC) \wedge (C) \tag{3.8}$$

Finally, from the four-node reduced diagram (nodes in Figure 3 with red marks), or equivalently Formulae (3.5)(3.6)(3.7)(3.8), we introduce priorities and construct the final overlapped table with priorities. The priority follows two rules:

- Nodes at lower level take precedence over nodes at upper level.
- For same-depth nodes with a common child, prioritize nodes sharing the child's action over others.

After introducing overlaps and removing negated terms, Table 7 is the final minimal policy table equivalent to Table 6 . After we did reduction for each snippet, for each key we can combine every snippet together to get the corresponding output sub-table.

For tables with multiple key-fields we can do reduction following the same steps outlined in this section. When we build reduction diagram, each node contains multiple fields.

4 Evaluation

Our evaluation aims to answer whether P4CGO can optimize P4 program for a given objective function and how effective our approach is.

We have implemented a prototype of P4CGO System in Java. The system takes generated control plane merged policy table as input, transforms the merged policy table into equivalent RegEx and builds DFA to accept the RegEx. It then enumerates all possible table splits on DFA and perform reduction algorithm to generate output tables. Finally, P4CGO finds the best table split by given objective function and output optimized control plane tables.

We obtained access control lists from Purdue network topology [18] as P4CGO policy inputs. Our sample includes 1592 access control list (ACL) tables. While the original rule tables are already small, we are not aware of other publicly available real-world datasets with a larger number of rules. In the future we plan to explore scalability using synthetic rule generators. We run P4CGO on a machine with Intel Xeon 3.10 GHz, 36-core CPU and 188 GB RAM.

4.1 Optimization

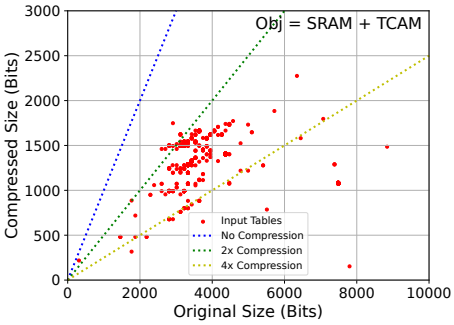
While developing P4 programs, programmer want to consider multiple optimization aspects such as a stage's SRAM and TCAM memory, number of stages, ALU usage, registers, etc. We want to support user defined objective function but do optimizations by restructuring control plane and data plane instead of reordering tables or reusing data structures in data plane. Given a merged policy table and objective function, P4CGO can perform an exhaustive search to find best splitting strategy.

To explore the optimizations that P4CGO can perform, we use ACL policy tables as input and apply user-defined objective functions to generate the desired control plane tables and data plane table sizes. Figure 4a illustrates a compression objective function where the objective value is defined as $Obj = TCAM + SRAM$. Under this objective we achieved an average compression rate of 63.83%. Users can also define custom objective functions such as $Obj = x \times TCAM + y \times SRAM + z \times STAGE$, or set constraints $STAGE < n$ to emphasize different aspects of optimization.

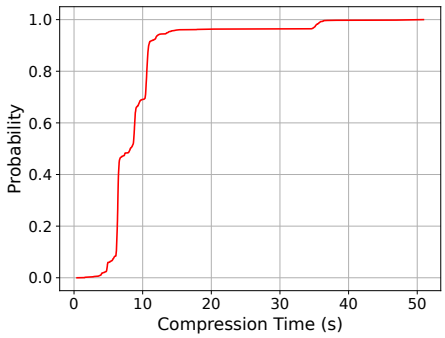
If we give TCAM a higher weight and set an objective function as $Obj = 0.25 \times SRAM + TCAM$, intuitively, we can focus more on saving TCAM memory when P4CGO aims to minimize the obj value. Table 8 demonstrates this idea by applying different optimization metrics to the same input ACL table. For this input ACL table, all *Ports* and *Protocols* are exact matches while *IPs* contain ternary values. As illustrated, emphasizing TCAM optimization leads to offloading as many keys as possible to SRAM, thereby minimizing the usage of TCAM memory.

4.2 Performance

P4CGO's effectiveness is significant because many networking systems require frequent policy updates and deploy tables on hardware. The effectiveness of P4CGO is decided by the number of key fields and number of entries of the merged policy table. Figure 4b demonstrates the effectiveness of our algorithm. For all input policy tables in §4.1 we have most of the tables optimized in 37 secs.



(a) Compression Rate



(b) Compression time

Figure 4: Compression Rate and Time of the ACLs on the Purdue Router Configuration Dataset

Table 8: Optimizations under Difference Objectives

Objective Function	Original Table Size/bit	Keys in SRAM Table	SRAM Table Number	SRAM Table Size/bit	Keys in TCAM Table	TCAM Table Number	TCAM Table Size/bit
SRAM + TCAM	8840	NA	0	0	SrcIP, SrcPort, DstIP, DstPort, Protocol	2	1495
0.25SRAM + TCAM	8840	Protocol, SrcPort, DstPort	3	653	SrcIP, DstIP	2	925

5 Related Work

Ever since OpenFlow introduced a new opportunity to the research community of computer networks and opened up the era of software-defined networks (SDN), it enabled software to control networks while also exploiting the fast process rate of switching hardware [11]. Although SDNs provided opportunities to deploy software on the control plane and make the data plane programmable, memory shortage issues remain, even with the widespread use of TCAM memory. Much previous work has focused on optimizing tables for performances while fitting within limited hardware capabilities. In recent years, optimization approaches have bifurcated into two directions: one direction is to introduce higher-level, more abstract languages paired with user-friendly programming tools [6, 8, 15, 16]; the other direction emphasizes the development of advanced optimization techniques that require user-provided

guidance [1, 17, 19]. Previous work [7] investigated a solution to reconfigure and compress the policies based on readability. More recently, researchers constructed geometric models for the policy compression problem [3].

SPliT [12] performs table compression by decomposing a d-dimensional table into $k \leq d$ smaller tables stored into a pipeline of k smaller TCAM chips. By *splitting* the policy they can overcome the multiplicative effect hence reduce the total required TCAM space. Different from P4CGO using symbolic encoding, SPliT transforms the policy table into a decision tree using ternary key's *numeric* values and cut the tree into k fields. It performs table merging by allowing multiple rules from different TCAM tables to co-reside in the same TCAM entry [13]. When rules between two tables co-reside, the compression depends on the level of commonality detected among the rules across both tables. In SPliT's working cases all overlapped *lpm* entries can be safely co-resided due to the contained relationship inherent in these overlaps. However when considering more general cases such as ternary or exact rules, guiding the compression by commonality might be misleading.

P4All [8] also aims to enhance the flexibility and efficiency of programming network switches under limited resource conditions. It emphasizes a modular design for P4 programs, allowing for the reusable data structures such as hash tables and hash-based matrices. To achieve this, P4All employs symbolic primitives to parameterize the size and shape of these structures and uses objective functions to quantify their values. While our approach also utilizes objective functions to guide optimizations, it does not focus on reusing data structures.

Researchers have also developed languages to program the whole distributed system. Frenetic [5] utilized a declarative query language to build a distributed network switch system that classifies and manages network traffic. Other functions of Frenetic included describing packet-forwarding policies and passing on packet-processing rules to its run-time.

6 Future Work

As a preliminary work, P4CGO has several limitations that could be addressed in the future to enhance its capability and applicability. The first one is scalability. In §3.1, P4CGO expands entries into disjoint *virtual* entries and this expansion is exponential. Although the expansion is handled in §3.3 by merging disjoint entries back into TCAM entries, the intermediate step (DFA Splitting in §3.2) still experiences exponential expansion. Another problem is that P4CGO optimizes control plane in a static criterion—a dynamically changed policy could result in a redeploy of generated optimal data and control plane. How to efficiently produce large scale table layouts that can accommodate the frequent updates is a key focus for future work.

7 Conclusion

We have presented P4CGO, a system using control plane to guide the data plane optimization. By performing table merging and enumerating table splitting, P4CGO can transform input policies into a provably optimal implementation for target hardware. We further perform experiments to show P4CGO's effectiveness in achieving optimal performance while maintaining policy compliance in practical network scenarios.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Award Nos. CCF-1837023, CCF-2046071, CCF-2319425. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Anubhavnidhi Abhashkumar, Jeongkeun Lee, Jean Tourrilhes, Sujata Banerjee, Wenfei Wu, Joon-Myung Kang, and Aditya Akella. 2017. P5: Policy-driven optimization of P4 pipeline. In *Proceedings of the Symposium on SDN Research* (Santa Clara, CA, USA) *(SOSR '17)*. Association for Computing Machinery, New York, NY, USA, 136–142. https://doi.org/10.1145/3050220.3050235

[2] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 87–95. https://doi.org/10.1145/2656877.2656890

[3] Yuzhu Cheng, Weiping Wang, Jianxin Wang, and Haodong Wang. 2019. FPC: A new approach to firewall policies compression. *Tsinghua Science and Technology* 24 (02 2019), 65–76. https://doi.org/10.26599/TST.2018.9010003

[4] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: an efficient SMT solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (Budapest, Hungary) *(TACAS'08/ETAPS'08)*. Springer-Verlag, Berlin, Heidelberg, 337–340.

[5] Nate Foster, Rob Harrison, Michael J. Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker. 2011. Frenetic: a network programming language. *SIGPLAN Not.* 46, 9 (sep 2011), 279–291. https://doi.org/10.1145/2034574.2034812

[6] Jiaqi Gao, Ennan Zhai, Hongqiang Harry Liu, Rui Miao, Yu Zhou, Bingchuan Tian, Chen Sun, Dennis Cai, Ming Zhang, and Minlan Yu. 2020. Lyra: A Cross-Platform Language and Compiler for Data Plane Programming on Heterogeneous ASICs. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 435–450. https://doi.org/10.1145/3387514.3405879

[7] Mohammad Hajjat, Xin Sun, Yu-Wei Eric Sung, David Maltz, Sanjay Rao, Kunwadee Sripanidkulchai, and Mohit Tawarmalani. 2010. Cloudward bound: planning for beneficial migration of enterprise applications to the cloud. *SIGCOMM Comput. Commun. Rev.* 40, 4 (aug 2010), 243–254. https://doi.org/10.1145/1851275.1851212

[8] Mary Hogan, Shir Landau-Feibish, Mina Tahmasbi Arashloo, Jennifer Rexford, and David Walker. 2022. Modular Switch Programming Under Resource Constraints. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 193–207. https://www.usenix.org/conference/nsdi22/presentation/hogan

[9] Yifan Li, Jiaqi Gao, Ennan Zhai, Mengqi Liu, Kun Liu, and Hongqiang Harry Liu. 2022. Cetus: Releasing P4 Programmers from the Chore of Trial and Error Compiling. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 371–385. https://www.usenix.org/conference/nsdi22/presentation/li-yifan

[10] Edward J. McCluskey. 1956. Minimization of Boolean Functions. *The Bell System Technical Journal* 35, 5 (1956), 1417–1444. https://doi.org/10.1002/j.1538-7305.1956.tb03835.x

[11] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.* 38, 2 (March 2008), 69–74. https://doi.org/10.1145/1355734.1355746

[12] Chad R. Meiners, Alex X. Liu, Eric Torng, and Jignesh Patel. 2011. Split: Optimizing Space, Power, and Throughput for TCAM-Based Classification *(ANCS '11)*. IEEE Computer Society, USA, 200–210. https://doi.org/10.1109/ANCS.2011.36

[13] Chad R. Meiners, Jignesh Patel, Eric Norige, Eric Torng, and Alex X. Liu. 2010. Fast regular expression matching using small TCAMs for network intrusion detection and prevention systems. In *Proceedings of the 19th USENIX Conference on Security* (Washington, DC) *(USENIX Security'10)*. USENIX Association, USA, 8.

[14] Willard V. Quine. 1952. The Problem of Simplifying Truth Functions. *Amer. Math. Monthly* 59, 8 (1952), 521–531. https://doi.org/10.2307/2308214

[15] John Sonchack, Devon Loehr, Jennifer Rexford, and David Walker. 2021. Lucid: a language for control in the data plane. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference* (Virtual Event, USA) *(SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 731–747. https://doi.org/10.1145/3452296.3472903

[16] Hardik Soni, Myriana Rifai, Praveen Kumar, Ryan Doenges, and Nate Foster. 2020. Composing Dataplane Programs with μP4 *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 329–343. https://doi.org/10.1145/3387514.3405872

[17] Nik Sultana, John Sonchack, Hans Giesen, Isaac Pedisich, Zhaoyang Han, Nishanth Shyamkumar, Shivani Burad, André DeHon, and Boon Thau Loo. 2021. Flightplan: Dataplane Disaggregation and Placement for P4 Programs. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, 571–592. https://www.usenix.org/conference/nsdi21/presentation/sultana

[18] Yu-Wei Eric Sung, Sanjay G. Rao, Geoffrey G. Xie, and David A. Maltz. 2008. Towards systematic design of enterprise networks. In *Proceedings of the 2008 ACM CoNEXT Conference* (Madrid, Spain) *(CoNEXT '08)*. Association for Computing Machinery, New York, NY, USA, Article 22, 12 pages. https://doi.org/10.1145/1544012.1544034

[19] Patrick Wintermeyer, Maria Apostolaki, Alexander Dietmüller, and Laurent Vanbever. 2020. P2GO: P4 Profile-Guided Optimizations. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks* (Virtual Event, USA) *(HotNets '20)*. Association for Computing Machinery, New York, NY, USA, 146–152. https://doi.org/10.1145/3422604.3425941

[20] Jiarong Xing, Yiming Qiu, Kuo-Feng Hsu, Songyuan Sui, Khalid Manaa, Omer Shabtai, Yonatan Piasetzky, Matty Kadosh, Arvind Krishnamurthy, T. S. Eugene Ng, and Ang Chen. 2023. Unleashing SmartNIC Packet Processing Performance in P4. In *Proceedings of the ACM SIGCOMM 2023 Conference* (, New York, NY, USA,) *(ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 1028–1042. https://doi.org/10.1145/3603269.3604882

Received 24 May 2024; accepted 7 June 2024