ShadowFS: Speeding-up Data Plane Monitoring and Telemetry using P4

Ricardo Parizotto, Lucas Castanheira, Rafael Hengen Ribeiro, Luciano Zembruzki
Arthur Selle Jacobs, Lisandro Zambenedetti Granville, Alberto Schaeffer-Filho
Institute of Informatics – Federal University of Rio Grande do Sul
Av. Bento Gonçalves, 9500 – Porto Alegre, Brazil
Email: {rparizotto, lbcastanheira, rhribeiro, lzembruzki, asjacobs, granville, alberto}@inf.ufrgs.br

***Abstract*—Programmable Data Planes (PDPs) provide software abstractions for network operators to dynamically modify the data plane behavior. This behavior can be described in specification languages, such as P4, and deployed into programmable switches our routers. The degree of innovation enabled by PDPs allowed network operators to create new protocols and applications. Despite the high degree of innovation brought to data plane packet processing, this programmability may have a negative effect on the forwarding delay and update times of flow tables. Previous works have attempted to overcome these limitations, e.g., through caching mechanisms, however they do not provide efficient replacement primitives and incur large overhead for monitored traffic. In this paper we present the design and evaluation of ShadowFS, a system to speed-up monitoring and telemetry on the data plane. ShadowFS manages the replacement of table entries using smaller caches without requiring the programmer to specify the behavior of these tables or how to steer traffic through them. Different from previous work, ShadowFS builds a new data plane program that monitors flows and replaces rules between tables automatically. Evaluation results demonstrate that ShadowFS can increase the throughput of frequently monitored flows.**

I. Introduction

Software-based paradigms for networking enable decoupling software solutions from the hardware in which they execute, making the management and operation of the network infrastructure more flexible and adaptive. Programmable switches present an alternative to traditional switches, by allowing the dynamic insertion of new functionalities into the network [1]. High-level programming languages, such as P4 [2], emerged to offer more flexibility in the development of protocols and network functionality by allowing packet processing at line rate in the switch itself. The utilization of programmable switches allows, for instance, the deployment of security mechanisms [3] or load balancers on demand [4], without requiring the network operator to acquire specialized hardware for these tasks.

Despite the fact that programmability enables more flexibility in terms of switch operation and packet forwarding, it also presents several challenges. Programmable switches can make the packet forwarding slower because of the usage of logical tables, as opposed to TCAMs which are used in traditional switches. This loss in performance occurs because packet lookup in logical tables takes longer than using TCAMs (which can do it in constant time), leading to low throughput rates and latency when the number of tables is large. Caching table entries may allow better usage of network resources without introducing significant overhead to the data plane [5]. However, traditional cache replacement mechanisms, such as LRU (Least Recently Used) and LFU (Least Frequently Used), do not consider important factors, and it is not clear how bandwidth consumption is affected by the cache replacement mechanism. Therefore, there is a need for mechanisms to cache forwarding rules of programmable switches without introducing significant overhead for packet processing. Yet, few works [5] have proposed replacement mechanisms in programmable data planes, and these are not targeted to monitoring and telemetry scenarios.

In this paper we present the design and evaluation of ShadowFS, a system to speed-up the monitoring of forwarding tables. ShadowFS relies on the utilization of a caching mechanism to increase the throughput of monitored flows. ShadowFS is built over our previous work, called FlowStalker [6], a two-phase monitoring scheme that runs directly on the data plane. ShadowFS enables the usage of the monitoring scheme through a new language operator proposed for P4. Dynamically, the system leverages monitored metrics to cache the most frequent entries in a faster table. The system shields the programmer from having to specify the behavior of these new tables or how replacement occurs.

ShadowFS is divided into two main components: the PreProcessor and the HotSwapper. The PreProcessor operates statically, extending P4 programs and dividing each forwarding table into two logically split tables. We call these the "shadow" and the "main" tables. The HotSwapper operates dynamically and transfers rules between the shadow and the main table. To this end, the HotSwapper has to efficiently perform monitoring and keep track of flows and packet counters of each table entry on the switch itself. It also needs to enforce a cache-replacement mechanism, which calculates heuristics and swap table entries between the monitored tables, always keeping the most utilized entries in the faster table. Our results demonstrate that ShadowFS can improve throughput of the most frequent flows and therefore improve the overall performance of a monitored system.

In the remainder of this paper, we provide a brief overview of Programmable Data Planes (PDP) (§II) and related work (§III). We explain the design of ShadowFS and its scope of

978-1-7281-5089-5/20/$31.00 ©2020 IEEE

operation (§IV), allowing the developer to easily instantiate tables which will behave with a cache. Finally, we present evaluation results to show the flexibility and trade-offs of ShadowFS (§V), and the concluding remarks (§VI).

II. Programmable Data Planes

Programmable switches have been utilized by network operators to modify how packets are processed in the data plane, without requiring them to buy specialized hardware to deploy each new functionalities. Usually, this kind of switch executes in general-purpose hardware, such as OpenvSwitch (OvS) [1] and PISCES [7], or programmable hardware, like Tofino switches [8] and netFPGA [9]. Programmability also enables the construction of robust testing and emulation platforms, such as Mininet and BMv2 [10], which are useful for the development of services and tools for SDN.

More recently, the development of specification languages such as P4 [2] enabled the configuration of the behavior of programmable switches without rewriting low-level instructions (e.g., the kernel of OvS, integrated circuits of hardware switches). The P4 specification language allows the programming and configuration of forwarding devices, including specific actions or control calls. In contrast to standard OpenFlow switches [11], P4 enables network developers to build programs that modify the structure of packet headers and can store complex network state on the data plane switches.

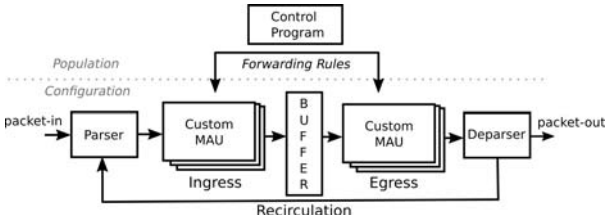

Fig. 1: The abstraction of P4, adapted from Bosshart et al. [2]

The P4 abstraction, depicted on Figure 1 divides the data plane behavior into three main blocks: the Parser, Control flows and the Deparser. The parser is a state machine that describes how to read headers from incoming packets. After a packet is processed by the Parser it follows to a processing pipeline. Each control flow is composed by a set of logical *match+action* tables implemented using *match+action units* (MAUs). An *apply* block specifies the semantics and order that each MAU processes packets and modifies the content of header attributes instantiated by the Parser. The Deparser writes internal variables to the packet header and emits the packet to an output port (or recirculates it back to the parser).

P4 also divides the forwarding model into two stages, the *configuration* and the *population*. During the *configuration*, developers can configure the parser state machine, the structure of *match+action* tables and the semantics of control flows. The *population* stage is when developers can insert, remove, or modify entries of the stateful objects (such as tables and registers). In the case of P4, the language does not dictate table update behavior, therefore it is necessary to build tools on top of P4 to provide an update command for a target switch, i.e., when a packet matches a rule, an action is invoked with parameters supplied by a control plane application.

III. Related Work

In this section, we present the main research efforts that attempt to optimize forwarding table operations, which is the main focus of this work. Hermes [12] presents a cache-based mechanism for traditional switches to decrease update time. Hermes provides a framework to give guarantees without requiring changes to the TCAM's fundamental architecture. The performance guarantees are made possible by the logical partition of TCAMs and the managing of the number of rules in each partition. Provably correct optimizations ensure that table entries are replaced when space heuristics indicate to do so. However, this mechanism is not suitable for general packet processors such as P4.

CacheFlow [13] caches the most frequent table rules in a smaller TCAM. The system architecture consists of TCAM and software switches, which can run on CPUs. OpenFlow rules of the software switches are then replaced by frequent rules on the TCAM. The system also solves dependencies between rules to reduce the number of table entries in the fast table. However, they only support OpenFlow switches. The integration of data plane programmability would require new mechanisms to support features such as tables with custom actions or packet headers.

CacheP4 [5] presents a cache mechanism for general packet processors. The system uses a single logical table to cache packet flows traversing a P4 switch. CacheP4 positions the cache in front of the entire pipeline to perform a lookup on packet header fields. The table contains the features of flows as well as the corresponding cache behavior. However, CacheP4 lacks mechanisms for replacing rules into the provided tables. This requires the network operator to perform the modifications of the rules manually.

B-Cache [14] proposes advances to behavior caching for PDPs. B-Cache uses one single additional table to cache all the pipelines. The system analyzes stateful elements of P4 programs, such as registers and intrinsic metadata, being able to detect hot behaviors. Primitives are then utilized to update the entries corresponding to hot behaviors in the cache. However, once objects on the pipeline do not necessarily have the same match type, all the system is constrained to this same match type and therefore becomes difficult to integrate. Furthermore, the pipeline may become too large, since all the pipeline has a cached "copy".

MatReduce [15] is a framework that reduces duplicated matches of data plane entries, therefore optimizing the P4 pipeline. MatReduce merges equivalent tables at compile-time, reducing resources consumed by the data plane. The system, then, dynamically translates user rules to the optimized pipeline. Despite the size of the table pipeline, it must incur even in higher overheads to update tables, since each update is delayed by the translation of rules. Additionally, MatReduce

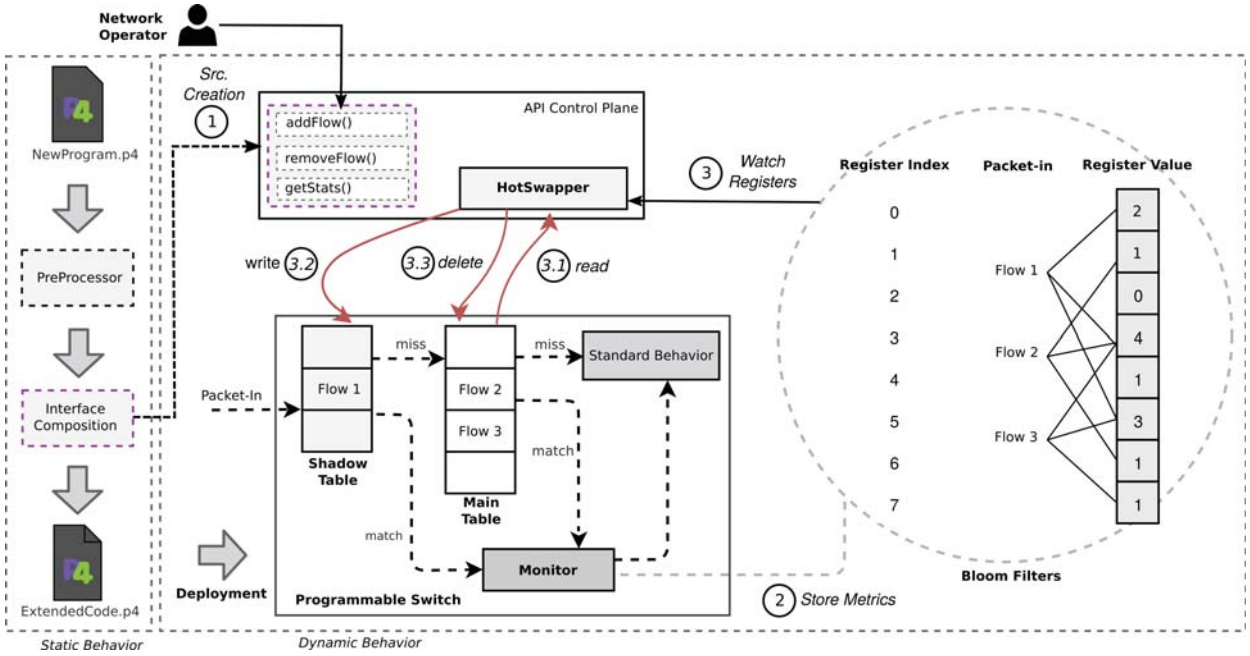

Fig. 2: Architecture of ShadowFS. Step 1: program PreProcessor splits forwarding tables and composes the control plane interface. Step 2: Each register of a bloom filter begins as an zero-filled array. The flow ID of a packet is hashed k times with each hash yielding a register index. Step 3: The HotSwapper reads contents of registers and transfers table entries using the replacement mechanism

may create large bunches of table entries, since the content of distinct tables is centralized at the same logical table.

IV. ShadowFS Design

The utilization of specification languages such as P4 for programming virtual switches avoids the need to statically rewrite the switch kernel manually to deploy a new data plane behavior. Although programmability may favor resource management, the forwarding becomes limited to resources of the host machine, which is usually slower than TCAMs of traditional switches. To speed up the monitoring of packet flows, we present ShadowFS, a cache replacement mechanism for programmable data planes using P4. The system employs additional data plane structures and replacement heuristics, aiming to reduce the delay and increase the throughput of flows.

ShadowFS is based on the premise that the lookup time on programmable switches is proportional to the number of entries on the tables. Tables with too many entries can make packet lookup slow and consequently slow down the main operations on forwarding tables. To overcome this problem, ShadowFS divides each P4 table into two different logical tables (the "shadow" [16] and the "main" tables), and moves the most utilized flows to the fastest table when necessary. ShadowFS works as an intermediary and transparent layer between P4 programs and the physical programmable data plane. The system is built over FlowStalker [6], a monitoring mechanism which encodes metrics and stores them on data plane devices for subsequent collection. Specifically, our proposed system monitors per-flow metrics (e.g., packet counts, timestamps), and uses this information to guide the replacement of table entries.

The overall view of ShadowFS is presented in Figure 2. ShadowFS is composed of two main subsystems: the PreProcessor and the HotSwapper. In the next sections, we present an in-depth description of each component.

A. Program PreProcessor

With the aim of making our system transparent to the data plane developer, the PreProcessor interprets the switch specification code and splits forwarding tables into two tables. Assuming that every table needs to be extended to operate jointly with a shadow could have an undesirable impact on the forwarding performance. Instead, we provide an extension for the P4 language to specify which tables should have a cache. The extension includes additional assertions on tables, which indicate that a table will behave with our cache mechanism and the performance bounding. The performance bounding is a parameter which dictates the shadow size, *e.g.*, if the operator wishes 50% of performance improvement for rules in the shadow, the shadow will have half the size of the main table and so on.

Figure 3 presents an example of how developers could indicate that a P4 table will have a shadow extension. In this example, the table `ipv4_lpm` would compile to a new table with only 80% of its original size.

Splitting tables. The PreProcessor is designed as an extension for the P4 development toolkit. It interprets P4 code

```
table ipv4_lpm {
  key = {
    hdr.ipv4.dstAddr: lpm;
  }
  actions = {
    ipv4_forward;
    drop;
  }
  size = 1024;
} has Shadow 80;
```
Fig. 3: Shadow tables are specified after the main declaration

and extends the code to integrate the cache mechanism into the switch specification. The PreProcessor splits the assigned lookup tables into two logically separated tables. We call these tables the "shadow" table and the "main" table. To this end, the PreProcessor parses the P4 code and creates an identical copy of each table indicated to have shadow, but with only the number of entries equivalent to the parameter specified in the original table. ShadowFS renames table IDs by concatenating the original name with the sufix *"shadow"*.

Next, the PreProcessor rewrites the control flow to position the new table in the switch pipeline. Cache tables are placed before the main table in the switch pipeline. If a packet hits a rule in the cache, it is processed by the respective actions and then follows to the standard behavior of the switch. Otherwise, i.e., if a packet lookup does not hit the cache, the packet lookup follows to the main forwarding table. After the PreProcessor deploys the new cache tables on the switch target, the HotSwapping starts running.

Shadow API. After parsing the P4 code, a programming interface for the control allows the network operator to insert, remove, and get statistics of table entries (Figure 2, Step 1). ShadowFS writes configuration files which store information of tables specified with caches. The configuration is then utilized by the P4 CLI programming interface to differentiate tables with a shadow and perform reads and writes in a manner transparent for the developer. The shadow table is also transparent to network operators, which can install and remove table entries as if there were only one table. In the background, the HotSwapper component performs the replacement of table entries dynamically.

B. Swapping Table Entries

Just as traditional caching mechanisms, ShadowFS must be able to maintain the most frequently hit forwarding rules in the shadow table to obtain low lookup times and reduce delay. To ensure this property, the HotSwapper performs periodical reads and writes between the main and shadow tables. The HotSwapper works as a hybrid layer that is responsible for keeping the most frequently utilized entries of tables into the smaller and faster table. Although the HotSwapper could benefit from executing directly on the data plane, this is not possible because of constraints of P4 itself. In particular, P4 does not allow to self-add table entries or loops, which are essential to perform the replacement. As such, the HotSwapper component executes in the control plane.

The HotSwapper works by monitoring per-flow statistics from switches and moving the table rules between shadow and main tables. For this, the HotSwapper has two different modules: (1) the monitor, which watches the per-flow frequency of matches and stores this information on the switch itself; and (2) the cache-replacement mechanism, which pulls monitored entries and transfers rules as necessary.

Monitoring. The monitoring module runs on switches and stores metrics for per-flow entries for the respective monitored table [17]. Frequency counters are deployed after each forwarding table. For the monitoring to occur in a lightweight manner, there is a need for fast indexing of flows and low space overhead. To this end, we store information on stateful registers using bloom-filters [18], as they are indexed in linear time and keep metrics with small collision probability. Two different filters are deployed after each forwarding table annotated for shadowing, which are responsible for storing two different metrics:

- *Hit frequency*: the system employs counting bloom filters to monitor the number of hits of each table entry. In Figure 2, Step 2 we illustrate the usage of counting bloom filters for flow frequency monitoring [19].
- *Timestamping*: Bloom-filters store the last timestamp of each hit of flow entries.

Each packet lookup that hits the table is then mapped to the set of the registers we allocated. The mapping of flows to registers is performed using three different hash functions (`crc16`, `crc32`, `crcCustom`). In the example of Figure 2, Step 2, packets from Flow 1 are mapped to registers 0, 3, and 5, respectively. These register values are incremented to further analysis. The HotSwapper then reads the contents of data plane registers and performs the respective hashing to identify the target flows and get the min of the three hash-count values. Using this calculation, we estimate how many packets could match each specific table entry. Next, the system swaps entries with a high probability of being hit until the next round to the Shadow table (which is smaller and has fewer rules than the main table).

C. Update Management

To perform the swapping without disrupting normal packet forwarding, ShadowFS must not create intermediary states while the system is transferring rules from the main table to the shadow table [20]. To avoid the creation of intermediary states, rule transfer must be performed in the following order:

Insertion. There is a need for swapping table entries when an insertion is required but the Shadow table is overloaded. Table entries with less priority (defined by the replacement policy) are hot-swapped with the main table to free space for the new entries. The replacement is made similar to LRU, i.e., the table entry with the least hit timestamp is replaced by the the most updated entry.

Deletion. ShadowFS only removes an entry from the tables it populates. But for the deletion to occur in a correct manner,

both deletion and update of table entries should be done atomically. For this, the update interface and the HotSwapper are defined as concurrent and must lock/free the ability to process these operations.

In Figure 2, Step 3, we present the sequence of steps the HotSwapper performs to avoid disruption: First, the control plane reads table entries from the data plane (Step 3.1). After calculating the replacement heuristics, the entries which are chosen to be in the shadow are inserted (Step 3.2). Now that the shadow is already processing packets for the new table entries, these can be finally deleted from the main table (Step 3.3).

V. Experimental Evaluation

In this section, we present evaluations that we performed to assess ShadowFS.

A. Metrics Formulation

Commonly, end-to-end measurement tools, such as `iperf` and `ping` considers information which are not useful to assess ShadowFS, as the system operates exclusively on the processing pipeline of the switch. Therefore we follow a different methodology [21] to measure latency. We need to assess only the latency of the control flows, as the time spent on parsing and deparsing is not important in this case. When a packet enters the pipeline and matches the steering table, we store a local timestamp, denoted as $Timestamp_i$. $Timestamp_i$ is stored as part of a packet state until the last program of the pipeline is concluded. On the last stage of the pipeline, $Timestamp_e$ is stored into a local register for further analysis. These correspond to ingress time and egress time, respectively.

Throughput represents the amount of data the switch can process in a given time. Similarly to what we did with the latency measurements, we do not consider parsing and deparsing time on the calculation of throughput. Thus, throughput is calculated as the effective throughput of processing a program's control flows p less the time spent on the ingress buffer b. Removing b is justified because it is not overhead of the additional tables or the replacement actions. Therefore we defined the throughput as $T = (p - b)/n$ where n is the number of packets traversing the switch.

B. Experimental Setup

The evaluation of our mechanism needs to measure both the impact of the monitoring and throughput for the usual flow processing through the shadows. For this, we observed the monitoring of annotated tables on the BMmv2 software switch. The experiments were performed on a Linux virtual machine with one logical CPU at 3.60GHz and 4GB of RAM. We configured a network scenario that performs a thousand requests/responses to measure latency and derive throughput.

Monitoring Overhead. Firstly, we measure the overhead of monitoring flows. The measurements were obtained from running test packets and tables with only one match entry. Figure 4 shows the throughput of forwarding packets, as we increase the number of monitored tables. In this experiment,

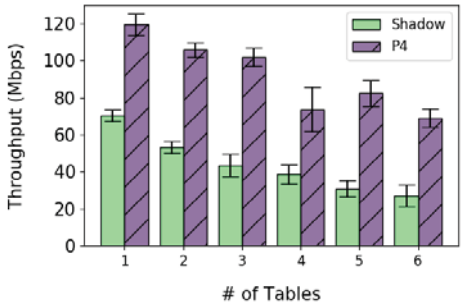

Fig. 4: Monitoring single flow rules

we observed that throughput increases with the number of watched tables. With six monitoring tables, throughput using shadow tables is almost half of the native P4 (with no monitoring). This justifies our design decision to place the shadow only in the assigned tables, instead of creating a new cache to every table or register. Specifying shadows as a language extension allows the developer to define which tables are more important for each application and would take more advantages of the caching behavior (those which will use more table entries).

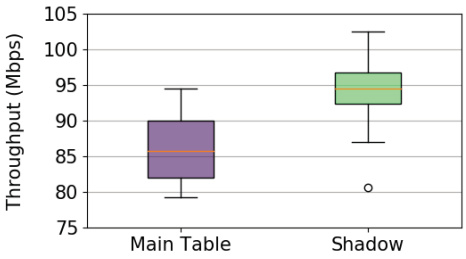

Fig. 5: Throughput with multiple flow rules

Caching Table entries. To asses the impact of ShadowFS on the forwarding of packets on general purpose hardware, we annotated the forwarding table of a simple L3 router. The main table has 1024 entries and the shadow only 128 slots. We filled the capacity of both tables and traversed them with a thousand packets through each table. Figure 5 presents results of the simulations. Packets which match the Shadow table have median throughput of approximately 95 Mbps, while the main table imposes a higher overhead, of approximately 85 Mbps. Therefore, ShadowFS improves the forwarding of monitored flows on the data plane.

C. Discussions

All experiments above are ideally based on a single source and destination flow traversing the P4 switch. However, multiple flows with different packet sizes would traverse the switch and consequently impact throughput. We see as future work developing the means to assess the impact of monitoring entries on more heterogenous scenarios.

The P4 behavioral model tables are implemented as hash tables, which make the insertion time constant. Tables implemented as TCAMs, however, have the insertion time linear to the number of entries. Therefore, insertions on the shadow may be faster than inserting on the main table. We advocate that the table division would improve the update performance on these hardware and still can be implemented on general purpose hardware using our mechanism.

Despite the overhead to forward packets created by the monitoring, the metrics from the monitoring can be utilized by analysis applications, to visualize performance patterns in real time or re-actively deploy changes in the data plane. In the case those metrics are required by the applications, ShadowFS can be utilized to improve the forwarding of monitored packets.

VI. Conclusions

In this paper we presented the design and evaluation of ShadowFS. ShadowFS is a system to speed-up the forwarding of monitored flows in programmable switches. The system allows developers to specify forwarding tables with new properties, which will behave with caches with tight bounds. The design of our system is based on Hermes [12], but differs from it by providing primitives for programmable switches and general packet processors. Additionally, we address fundamental needs of general packet processors, enabling frequent entries to be processed with low latency and higher throughput.

We see as an exciting future work to perform evaluations using programmable hardware. We see also as future work employing algorithms to merge similar table entries, thus reducing the number of rules in tables. Solving conflicts between the rules of two split tables can also allow us to have error-free configurations. Further, we see as future work caching tables using different devices. This would allow us to also share the loading between nodes of the infrastructure and guarantee that end-to-end performance and resource constraints are ensured by the system.

Acknowledgments

We would like to thank CNPq for research grants 407899/2016-2 and 312091/2018-4. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, and also by NSF CNS-1740911 and RNP/CTIC (P4Sec) grants.

References

[1] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch," in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, 2015, pp. 117–130.

[2] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2656877.2656890

[3] Â. C. Lapolli, J. A. Marques, and L. P. Gaspary, "Offloading real-time ddos attack detection to programmable data planes," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2019, pp. 19–27.

[4] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "Hula: Scalable load balancing using programmable data planes," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '16. New York, NY, USA: ACM, 2016, pp. 10:1–10:12. [Online]. Available: http://doi.acm.org/10.1145/2890955.2890968

[5] Z. Ma, J. Bi, C. Zhang, Y. Zhou, and A. B. Dogar, "Cachep4: A behavior-level caching mechanism for p4," in *Proceedings of the SIGCOMM Posters and Demos*. ACM, 2017, pp. 108–110.

[6] L. Castanheira, R. Parizotto, and A. E. Schaeffer-Filho, "Flowstalker: Comprehensive traffic flow monitoring on the data plane using p4," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[7] M. Shahbaz, S. Choi, B. Pfaff, C. Kim, N. Feamster, N. McKeown, and J. Rexford, "Pisces: A programmable, protocol-independent software switch," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016, pp. 525–538. [Online]. Available: http://doi.acm.org/10.1145/2934872.2934886

[8] "Barefoot whitepaper: The world's fastest and mostprogrammable networks," 2016. [Online]. Available: https://barefootnetworks.com/resources/worlds-fastest-most-programmable-networks/

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

[10] "Behavioral model." [Online]. Available: https://github.com/p4lang/behavioral-model

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[12] H. Chen and T. Benson, "Hermes: Providing tight control over high-performance sdn switches," in *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '17. New York, NY, USA: ACM, 2017, pp. 283–295. [Online]. Available: http://doi.acm.org/10.1145/3143361.3143391

[13] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Cacheflow: Dependency-aware rule-caching for software-defined networks," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '16. New York, NY, USA: ACM, 2016, pp. 6:1–6:12. [Online]. Available: http://doi.acm.org/10.1145/2890955.2890969

[14] C. Zhang, J. Bi, Y. Zhou, K. Zhang, and Z. Ma, "B-cache: A behavior-level caching framework for the programmable data plane," in *2018 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2018, pp. 00 084–00 090.

[15] X. Chen, D. Zhang, and H. Zhou, "Matreduce: Towards high-performance p4 pipeline by reducing duplicate match operations," in *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018, pp. 1–7.

[16] R. Alimi, Y. Wang, and Y. R. Yang, "Shadow configuration as a network management primitive," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 111–122, Aug. 2008. [Online]. Available: http://doi.acm.org/10.1145/1402946.1402972

[17] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '17. New York, NY, USA: ACM, 2017, pp. 164–176. [Online]. Available: http://doi.acm.org/10.1145/3050220.3063772

[18] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[19] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet mathematics*, vol. 1, no. 4, pp. 485–509, 2004.

[20] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '12. New York, NY, USA: ACM, 2012, pp. 323–334. [Online]. Available: http://doi.acm.org/10.1145/2342356.2342427

[21] H. T. Dang, H. Wang, T. Jepsen, G. Brebner, C. Kim, J. Rexford, R. Soulé, and H. Weatherspoon, "Whippersnapper: A p4 language benchmark suite," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '17. New York, NY, USA: ACM, 2017, pp. 95–101. [Online]. Available: http://doi.acm.org/10.1145/3050220.3050231