Hermes: Providing Tight Control over High-Performance SDN Switches

Huan Chen
University of Electronic Science and Technology of China and Duke University

Theophilus Benson
Brown University

ABSTRACT

SDN controllers demand tight performance guarantees over the control plane actions performed by switches. For example, traffic engineering techniques that frequently reconfigure the network require guarantees on the speed of reconfiguring the network. Initial experiments show that poor performance of Ternary Content-Addressable Memory (TCAM) control actions (e.g., rule insertion) can inflate application performance by a factor of 2×! Yet, modern switches provide no guarantees for these important control plane actions – inserting, modifying, or deleting rules.

In this paper, we present the design and evaluation of Hermes, a practical and immediately deployable framework that offers a novel method for partitioning and optimizing switch TCAM to enable performance guarantees. Hermes builds on recent studies on switch performance and provides guarantees by trading-off a nominal amount of TCAM space for assured performance. We evaluated Hermes using large-scale simulations. Our evaluations show that with less than 5% overheads, Hermes provides 5ms insertion guarantees that translates into an improvement of application level metrics by up to 80%. Hermes is more than 50% better than existing state of the art techniques and provides significant improvement for traditional networks running BGP.

CCS CONCEPTS

• **Networks → Programmable networks**; **Network management**; **Bridges and switches**;

KEYWORDS

Software-defined Networking; TCAM Update; Performance

ACM Reference format:
Huan Chen and Theophilus Benson. 2017. Hermes: Providing Tight Control over High-Performance SDN Switches. In *Proceedings of CoNEXT '17, Incheon, Republic of Korea, December 12–15, 2017,* 13 pages.
DOI: 10.1145/3143361.3143391

1 INTRODUCTION

Software-Defined Networking offers flexibility and programmatic control over the network. However, this programmatic control requires frequent modifications of the network's forwarding tables (TCAM). For example, traffic engineering SDN control programs, e.g., Google's B4 [40] or Microsoft's SWAN [39], require frequent network reconfigurations to improve network performance. Similarly, service chaining SDN control programs [22, 35] require fast reconfiguration to ensure network correctness.

Unfortunately, while SDN promises fine-grained control over the network, current SDN switches use traditional hardware and software algorithms that are designed to support legacy protocols, e.g., BGP, OSPF, or MPLS. [1] These traditional techniques are barely effective for supporting frequent TCAM modifications [11] and are ill-suited for the high TCAM modifications demands of emerging SDN applications [22, 35, 40]. As a result of this mismatch, running modern SDN control programs on these SDN switches can significantly degrade the performance of the applications. Our experiments in Section 2, demonstrate that due to inefficiencies in the switches, the TCAM installation time can reduce application performance by a factor of 2×!

In this paper, we define control plane actions as the set of SDN control events/messages that the SDN controller uses to configure the switch's TCAM, e.g., OpenFlow's *FlowMod* operator used to insert, delete or modify rules in the forwarding tables.

Existing approaches [37, 43, 51, 62] seek to minimize TCAM insertion times, mask TCAM insertion latencies, or design software algorithms to work around these hardware issues. These approaches provide a best-effort attempt to minimize TCAM insertion latency. Since these approaches attack the symptoms (insertion latency) and not the root-cause (TCAM behavior) they mitigate and not eliminate the issues — large variations in switch performance still exists.

Unfortunately, without providing concrete performance guarantees for control plane actions, modern SDNs are unable to effectively support the growing number of novel use cases — critical infrastructures, cellular infrastructures, security systems, and virtual networks. For example, in 4G and 5G networks, there is a need to instantiate VoLTE connections within a predefined amount of time. Similarly, for cyber-physical systems [23], there is a need for networks that make strong performance guarantees. Only by redesigning switch software and algorithms to explicitly support frequent control plane actions (with performance guarantees) can SDN support emerging SDN control programs.

In this paper, we present Hermes, a framework for providing performance guarantees for control plane actions. Hermes builds on the observations [37, 42, 43] that control plane actions are expensive when a flow table contains a large number of entries.

To this end, Hermes eliminates large tables by carving a monolithic TCAM table into two tables: the first, a small table (in size),

CoNEXT '17, Incheon, Republic of Korea

978-1-4503-5422-6/17/12…$15.00
DOI: 10.1145/3143361.3143391

[1] P4 reuses traditional TCAM and provides little control over the TCAM management techniques that impact control plane action speeds.

that's called the *shadow table* and kept relatively empty. This small table services all insertion/modification requests, thus from the perspective of these requests the TCAM is small and mostly empty. The second, the *main table*, is a full sized table. As the shadow table grows in size, Hermes reduces its size by migrating entries from the shadow table to the main table. By controlling the size of the shadow table, Hermes provides the lowest average and tail insertion times of all available systems [43, 51].

The design of Hermes faces three key challenges. First, developing a framework that provides guarantees without requiring fundamental changes to the TCAM's design. To this end, we examined SDKs for modern Application Specific-Integrated Circuits (ASICs) and observed that we are able to support Hermes with existing TCAMs (§ 6). Second, simultaneously providing performance and correctness guarantees. We introduce a prediction algorithm to maintain invariants over the shadow tables and a TCAM orchestration algorithm that uses provably correct optimizations to partition rules and ensure correctness (§ 4 and § 5). Lastly, we design a set of APIs to enable network operators to request performance guarantees and understand the trade-offs between performance and TCAM overheads. Hermes tackles this challenge by presenting a simple API for making requests.

To demonstrate the benefits and explore the limitations of Hermes, we conduct microbenchmarks and evaluate Hermes using large-scale simulations with realistic topologies, traffic traces, and application workloads from data centers [9] and ISP networks [1, 10, 19]. Our evaluations show that Hermes is able to provide performance guarantee, improve rule installation, improve network update times and improve application performance. Additionally, we observe that the benefits of Hermes are more pronounced when the workloads require frequent network updates (modifications) or where RTTs are small (e.g. in the data center). Our evaluations show that with less than 5% overheads (in TCAM space), Hermes provides a 5ms insertion guarantee that translates into an improvement of rule installation time by 80% to 94%.

2 MOTIVATION

In this section, we present the case for Hermes in SDNs (§ 2.2) and traditional networks (§ 2.3) and discuss the characteristics of TCAMs that inspire the design of Hermes (§ 2.1).

2.1 TCAM Background and Measurements

TCAM is a fixed size memory structure capable of very fast lookups for a key with flexible structure [8] – a lookup into a TCAM table returns an entry that matches the key. To support multiple keys, e.g., longest prefix match (LPM) and VLAN, vendors partition a TCAM into multiple logically disjoint slices, with each slice configured to support lookups for a specific key [6, 12, 20]. Moreover, TCAMs are designed with proprietary algorithms and hardware to ensure that lookups in each slice are constant.

Although lookups are constant time operations, TCAM modification actions (e.g., insertions or modifications) incur variable time because of the TCAM hardware which shifts/moves rules. The TCAM stores entries in a list and each new entry must be inserted into a specific location within the TCAM to preserve correctness and maintain priorities. Thus to perform an insertion, the TCAM may have to move existing entries to make room for the new entry. The insertion time is a function of the time to perform this move which is proportional to the number of entries that must be moved and the cost of moving each entry.

Pica8 P-3290			Dell 8132F		
ASIC	Table Occupancy	Update/s	ASIC	Table Occupancy	Update/s
108 KB Firebolt-3	50	1266	54 KB Trident+	50	970
	200	114		250	494
	1000	23		500	42
	2000	12		750	29

Table 1: Rule Update Rate [42].

2.1.1 Measuring Control Plane Action Performance. Recent studies [38, 42, 43] of SDN switches and TCAM performance have analyzed the performance profiles of current SDN switches. Here we summarize their key findings and use them to help motivate Hermes's design choices.

Insertion time grows linearly with the number of rules: Measurements show that a switch's insertion behavior is a function of several features:

First, the priorities of the rules being inserted impacts the flow insertion time [38, 43]; rules with priorities are five times slower than rules without priorities. Furthermore, the order of insertion is important. For example, in some switches installing rules in ascending order of priorities is ten-times faster than descending order.

Second, the number of rules in the flow tables impacts the flow insertion time [42]. For example, with a Dell 8132F switch, inserting a rule in a flow table with 250 rules can be more than 10 times faster than in a flow table with 500 rules in it. In Table 1, we present the rule update rate for different flow table occupancy levels for different switches.

Third, the performance characteristics vary from switch to switch and the configuration of Hermes must be adjusted to account for these fundamental differences [38, 43]. For example, as observed in Table 1, with 50 entries already in the table, the Pica8 switch can support approximately 1266 updates per second where are the Dell switch can support a lower rate of 970 rates per second: more than 23% difference in performance.

Deletion is a simple and fast operation: Unlike flow insertions, flow deletions exhibit a relatively trivial and fast performance characteristics because rule deletions remove entries and they do not always require moving entries around. Specifically, rule deletion latency is independent of rule priority [38, 43] and is much faster than rule insertion [26].

Modifications, surprisingly, can be constant: Modifications require changing a rule's match or actions – for either, modifications are cheap and fast because they do not require moving TCAM entries. For example, "modifying 5000 entries could be six times faster than adding new flows" rules [43]. Alternatively, modifications that alter the priority of a rule may require moving TCAM entries and are fundamentally similar to deleting the existing entry and inserting the modified entry.

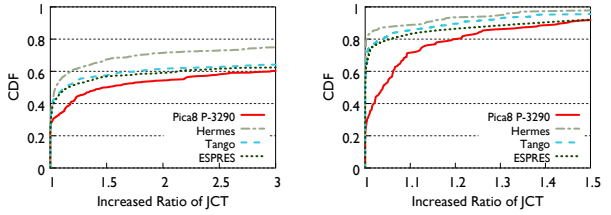
Takeaways The insertion time is directly proportional to the number of rules already in the flow table – we can bound the insertion time by limiting the number of rules in the flow table. Further, there is a clear correlation between the flow table size and the max insertion time (Table 1). Finally, while there are many types

of control plane actions only a few of them need to be revisited to provide strong performance guarantees. For example, flow table insertion, deletion, and modification are all part of the same control plane action, *flow-mod*, yet we only need to explicitly design for insertions.

2.2 Impact of Control Plane Action Latency on SDNs

In this section, we analyze the impact of control plane action latency (and variation) on networked applications (e.g., Map Reduce). To do this, we developed a flow-level network simulator, called Varys, based on existing flow-level simulators [29, 30] and modified it to support variable control plane action latencies. Our simulator emulates a k=16 Fat-Tree topology with 1024 servers and employs a proactive traffic engineering application [33] that monitors the network and periodically reconfigures the network paths to place flows on a more efficient network path — where an efficient path is defined as a path that minimizes congestion and job completion times. This proactive application does not use packet-in messages thus there is no startup latency incurred.

Our simulation modeled several switches based on empirically derived performance models [42] but due to space restrictions, we only provide results for the Pica8 P-3290 switch and note that the other switches have qualitatively similar results. These switch models allow us to model TCAM performance, both, control plane actions (rule installation/deletion/modification), data plane forwarding (packet matching and forwarding latencies), and TCAM behavior (TCAM shifting latency). The details of our simulator and workloads are presented in Section 8.



(a) Short Jobs (b) Long Jobs

Figure 1: CDF of Increase Ratio of JCT.

Figure 1 presents the impact of TCAM control latency and variation on Job Completion Times (JCT). We compare switches with no control plane latency (zero latency and variation) against switches with realistic control plane latency [42]. We separate short jobs, or jobs with less than 1 GB, (Figure 1 (a)) from long jobs (Figure 1 (b)). We observe that, compared to long jobs, short jobs are significantly impacted because short jobs are unable to ameliorate the added latency of the TCAM control plane actions: short jobs experience a 1.5× or 2× increase and long jobs experience a 1.05× or 1.25× increase in the median case. This trend is particularly alarming as the short jobs are often latency-sensitive and are arguably the more important flows [30].

For reactive applications which send the first packet to the controller and thus incur startup latencies, our evaluation in our prior work [28] showed that the control plane action latency can further reduce performance by a factor of 5×!

Compared with the alternative approaches, ESPRES [51] and Tango [43], Hermes provides significant improvements on job completions times (we elaborate on this in Section 8).

2.3 Impact of Control Plane Actions Latency on BGP

Next, we evaluate the frequency with which control plane actions are used within traditional BGP-based networks. To do this, we examine BGP updates captured by BGPStream [5] and convert the updates into FIB actions based on BGP's internal algorithms. Due to space constraints, we do not show the exact CDF of the update rates for these BGP routers and the CDF of performance improvements. We observe that traditional control planes generally have low update rates except at the tail where updates occur with high frequency (over 1000 updates per second) and it under these extreme conditions that traditional routers fail to react quickly enough because of the slow TCAM behavior. Poor TCAM behavior under these conditions has motivated the community [11] to call for proposals such as Hermes.

2.4 Future of SDN Hardware

Given the recent development of SDN-optimized switches, a fair question is this: "will the recent development of programmable data planes [4, 24] and switches optimized for SDNs [4, 14] eliminate variation and tail latency in TCAM control plane actions or obviate the need for TCAM?" Unfortunately, TCAM control latency (and variation) arises due to the properties that are fundamental to the design of TCAM memory. Thus, provided that TCAMs are used, variable latency will persist.

Due to the importance of the TCAM, emerging programmable switches, e.g., Barefoot's Tofino Chip [4], and programmable data plane models, e.g., Banzai [56] or OPP [25], include TCAM tables. Additionally, the P4 specification [17] provides no primitives for expressing performance requirements over TCAM control actions and subsequently, the P4 compiler lacks algorithms for managing and reconfiguring TCAM in a manner that provides performance guarantees.

While emerging specifications fail to address control plane action latency, there is a growing number of data plane techniques being developed that use TCAM and SRAM in a clever manner to provide advanced network functionality [24, 47, 48, 64] in commodity devices.

In summary, our results further highlight the need for systematic support for performance guarantees on control plane action latency. While existing approaches [41, 43, 45, 51] reduce the impact of TCAM control actions by reordering rules or designing new hardware algorithms, they fail to provide guarantees and this limits their applicability. Our work, Hermes, focuses on designing an efficient and robust solution that provides these guarantees on commodity switches.

3 ARCHITECTURE

At a high level, Hermes *bounds insertion time by restricting the size of the flow table.* To do this, Hermes maps a logical TCAM-based

flow table into two physical flow tables: the first table, a small table (the shadow table), that's kept relatively empty and the second table (the main table), a large table. All insertions happen in the shadow table while lookups happen using both tables sequentially; first, a lookup is performed into the shadow table and if no match is found then another lookup is performed into the main table. Together, these two tables provide equivalent functionality to a single (logical) table — Hermes includes a set of provable correct algorithms, based on ACL-optimizations, that enable Hermes to map rules from a logical table to two physical tables (the shadow and main tables) in a manner that simultaneously guarantees functional correctness (Section 4) and bounds performance (Section 5). During packet lookups, Hermes maintains the logical abstraction by configuring the default table-miss behavior in the shadow table to "forward to next table", in Hermes the next table after the shadow table is always the main table. This default behavior ensures that during packet lookup, the packet traverses the pipeline from shadow to main until it reaches a rule that processes it or until it traverses the entire pipeline. More broadly, as long as a matching rule exists in at least one of the two tables, the packet will be processed by this matching rule.

To provide bounds on TCAM insertion times, Hermes bounds the size of the shadow table and only inserts new rules into this bounded table. Note, the size of the shadow table bounds the maximum number of TCAM shifts that are required to insert a rule and, in turn, bounds the maximum insertion latency.

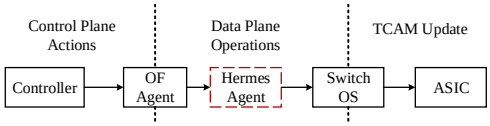


Figure 2: Level of Hermes.

Architecture. Hermes's architecture, Figure 3 (a), consists of two key components, *Gate Keeper* and *Rule Manager*. Hermes runs as a software agent on the switch intercepting TCAM related calls between the SDN switching agent (OpenFlow agent) and the switch ASIC drivers (Figure 2).

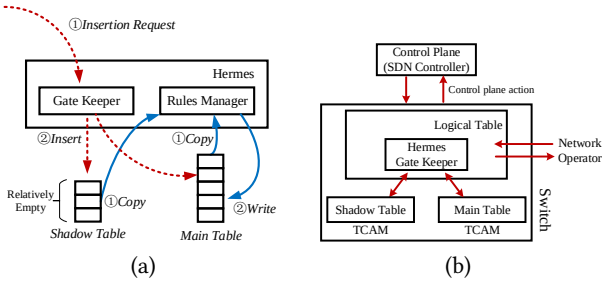


Figure 3: Architecture of Hermes (Single Table).

The Gate Keeper handles *flow-mod* actions and inserts rules into the shadow or main tables, depending on the rules and the configuration of Hermes (dashed red arrows in Figure 3). The Gate Keeper applies a set of predicates to the rule to determine if Hermes is configured to provide performance guarantees – if Hermes is configured then the shadow table is used else the main table is used. Additionally, the Gate Keeper maintains a token bucket that is used to ensure that the controller does not send actions faster than Hermes has agreed to make guarantees for. When the controller sends actions faster than the guaranteed rate, Hermes uses the main table to service the additional commands over the approved rate.

The Rule Manager periodically migrates (or copies) the rules from shadow table to main table (solid blue arrows in Figure 3). The goal of the Rule Manager is to ensure that rules are migrated from the shadow table before the occupancy of the shadow table exceeds its size and as long as the shadow table does not exceed its size, Hermes will provide its performance guarantees.

The primary challenges in realizing Hermes are:

Efficiency: TCAM tables are precious and scarce switch resources [18], Hermes must be carefully configured to ensure that the size of the shadow table is optimized to simultaneously provide performance guarantees while minimizing overheads.

Correctness: To ensure that the shadow and main tables, together, accurately provide identical functionality as a single logical table, Hermes must ensure that overlapping rules are inserted in a manner that respects priorities while preserving correctness. To this end, Hermes uses an efficient data structure to detect overlapping rules and selectively rewrite these rules to eliminate overlaps. Our approach is provably efficient and correct (Section 4).

Guarantees: Performance guarantees are central to Hermes's design. To this end, Hermes must maintain relatively empty shadow tables. Hermes includes a prediction algorithm that anticipates workload demands and preemptively migrates rules from the shadow to the main table — in essence defragmenting the TCAM to compartmentalize the rules (Section 5).

Practicality: Our current model is designed for switches with a single TCAM table. Yet, emerging SDN switches are complex and contain multiple TCAM tables. To this end, we discuss methods for extending Hermes and demonstrate the feasibility of implementing Hermes with commodity devices (Section 6).

Generality: Different generations of TCAMs provide different performance characteristics and applications require different guarantees. Hermes must learn and adjust to such heterogeneity. To this end, we build on existing vendor interfaces for configuring TCAMs [20] and introduce an interface (between the network operator and switch) to allow the operators to express guarantees and understand the overheads associated with these guarantees (Section 7).

Operational Workflow. At a high level (Figure 3 (b)), the network operator interacts with Hermes through the Gate Keeper's APIs (§ 7) to configure Hermes and request performance guarantees for control plane actions. Hermes uses these requests to determine the size of the shadow table and to, in turn, configure the switch's TCAM.

During normal operations, the control plane interacts directly with Hermes's agent which includes the Gate Keeper and Rule Manager functionality required to insert rules into the shadow table and migrate (or copy) rules into the main table.

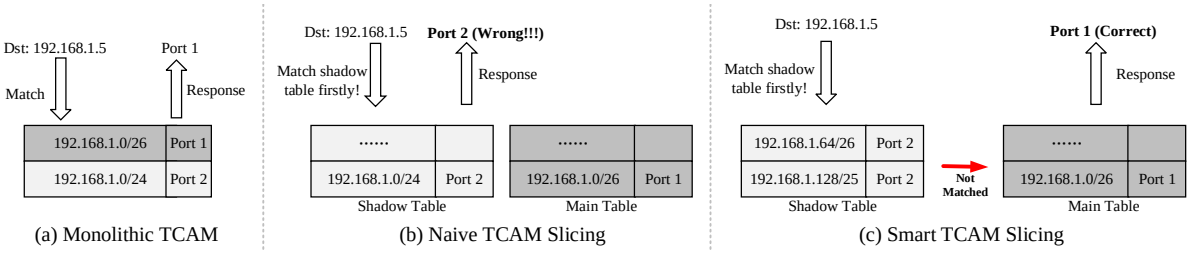


Figure 4: TCAM Correctness Violation.

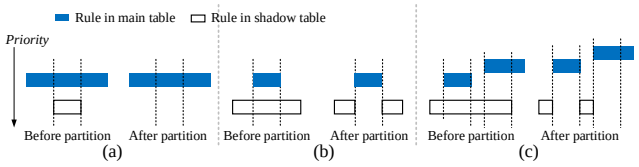


Figure 5: Rule Overlap.

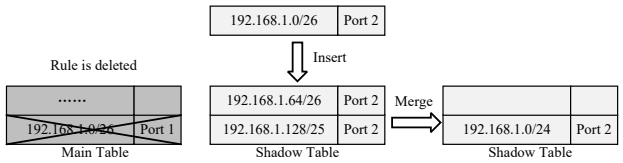


Figure 6: Ensuring Correctness after Deletion.

4 CORRECTNESS GUARANTEES

Hermes makes the following correctness guarantee:
The two tables maintained by Hermes will behave in an identical matter as a single monolithic table.

In this section, we describe our model of TCAM behavior and the algorithms used by Hermes to ensure correctness. When overlapping rules with different priorities are inserted into a TCAM table, the TCAM guarantees that the actions associated with the matching rules with the highest priority are applied to the packet. Consider the example in Figure 4. Two rules are inserted. The first rule, a higher priority rule, which forwards packets to port 1 (depicted in dark gray) and the second rule, a lower priority rule, which forwards packets to port 2 (depicted in light gray). In a monolithic TCAM table, (Figure 4 (a)), a lookup for a packet destined to "192.168.1.5" will be forwarded to port 1 because of the higher priority rule.

Correctness Violation During Insertion: In a naive implementation of Hermes, if the initial higher priority rule is migrated to the main table and the second, lower priority, rule is subsequently installed into the shadow table (Figure 4 (b)), then a lookup will return the wrong action ("forward to port 2"). Recall, Hermes examines the shadow table first and then examines the main table if and only if a table miss occurs in shadow table. The result of Hermes's lookup conflicts with the results from the traditional monolithic table — this is an obvious correctness violation.

When examining the implications of rule overlap, there are three conditions of interest which are shown in Figure 5: (a) the main table contains a larger, higher priority rule that wholly subsumes the new rule – to ensure correctness, the rule should be ignored and not inserted into the shadow table [2], (b) the main table contains a smaller higher priority rule that is subsumed by the new rule – to ensure correctness, the new rule should be partitioned in such a way that packets will still use the main table for the old rule, and (c) the main table contains one or more higher priority rules that overlap with the new rule in one or more locations – to ensure correctness, the new rule should be recursively partitioned [3]. Note, the example in Figure 4 falls into the category depicted in Figure 5 (b).

Algorithm 1: PartitionNewRule

input : New rule prefix r_{new}, existing main table rule prefix set E
output: Partitioned new prefix set N. Mapping set M.

```
1 O = φ,;
2 for prefix r ∈ E do
3     if Prio(r_new) < Prio(r) & DetectOverlap(n, r) = True then
4         O ← r
5 for prefix o ∈ O do
6     Partitioned prefix set P = EliminateOverlap(r_new, o);
7     Merge(N, P);
8 M ← {r_new, N};
9 return N, M
```

Correctness Violation During Deletion: Deleting a rule in the main table can impact correctness because the partitioned rules rely on the rules within the main tables to remain correct. For example, in Figure 6 if the rule in the main table is deleted then the partitioned rules no longer correct: lookups for 192.168.1.0/26 will not match any rules in either table.

4.1 Preserving and Enforcing Correctness

To enforce correctness, Hermes runs three algorithms, one during rule insertion to partition rules, another one during rule deletion to ensure that deletion does not impact correctness, and a last during modifications.

Rule Insertion: During insertion of a new rule, r_{new}, Hermes uses Algorithm 1 to examine and, if needed, partition the rule. Algorithm 1 has three high level steps: (i) detecting overlaps between the new rule, r_{new}, and all rules, $r \in E$, in the main table using ACL optimization functions [59] (Algo 1 line #3); (ii) then eliminating overlaps by iteratively cutting the new rule, r_{new}, into a set of partition rules, P, until no overlaps exist between any rule in P and any of the rules in the main table (Algo 1 line #6); (iii) finally, the

[2] This rule is redundant and would not be matched in monolithic table.

[3] (b) and (c) can be transferred to (a) by iteratively cutting and merging the rules

partitioned rules in P are merged using an optimal algorithm [59] to minimize the number of rules inserted into the shadow table . Algorithm 1 focuses on eliminating overlaps between the new rule and rules in the main table, however, overlaps between the new rule and rules in the shadow table are okay because we are inserting the new rule into the shadow table, and the underlying TCAM hardware correctly disambiguates overlapping rules provided that they are in the same table.

Rule Deletion: During deletion of a rule, r_{delete}, Hermes checks to see if the rule is in the shadow or the main table. For rules in the shadow table, Hermes checks to see if r_{delete} was partitioned by Algorithm 1. If r_{delete} was partitioned, then Hermes deletes all partitions; however, if r_{delete} wasn't partitioned Hermes deletes r_{delete}. [4] If r_{delete} is in the main table, Hermes deletes r_{delete} and checks to see if Algorithm 1 created any partitions in the shadow table due to overlaps with r_{delete}. If such partitions exists, Hermes appropriately "un-partitions" these rules by deleting the partition rules and adding back the original rule (Figure 6). To do this, Hermes maintains a mapping between the original rules and the partitioned rules (set M in Algorithm 1).

Rule Modification: During modification, if the priorities do not change, the modification is applied directly to the intended rule(s). Else, if priorities are being changed, the modification is converted into two actions: a delete of the original rule and an insertion of a rule capturing the "modification".

4.2 Rule Insertion Optimization

In certain situations, inserting into the main table does not trigger TCAM reordering or incur significant latencies. Specifically, when the new rule being inserted is the lowest priority rules [43]. In these situations, the Gate Keeper inserts the rule directly into the main table. This optimization has the additional benefit of minimizing the number of partitions because the rules that are more likely to be fragmented into a large number of partitions are these lower priority rules. [5]

5 PERFORMANCE GUARANTEES

Hermes provides performance guarantees by periodically migrating rules from the shadow table to the main table to keep the shadow table empty. Hermes's Rule Manager use a predictive algorithm to proactively infer the growth of the shadow table and trigger the migration of the rules from the shadow to the Main table. The design of the Rule Manager must tackle several challenges: (1) When to migrate rules from the shadow table? (§ 5.1) (2) How should the rules be migrated to maintain consistency? (§ 5.2) (3) Which guarantees does Hermes maintain during rule migration? (§ 5.2) Next, we elaborate on these challenges.

5.1 When to migrate rules?

To provide performance guarantees Hermes must migrate entries from the shadow table before it exceeds its capacity. This requires an intelligent technique for capturing and predicting the flow insertion rates. Given such a technique, Hermes can determine when shadow tables will exceed their thresholds and subsequently trigger a migration.

There are broadly three alternatives:

- **Application-Driven:** Modifying the controller interface and rewriting the SDN control programs (or SDNApps) to explicitly indicate their insertion rates to Hermes [43]. With such hints, Hermes can trigger migration when the hints indicate an overflow of the shadow table. This requires rewriting both the SDNApps and controller platforms.
- **Simple-Threshold:** Hermes can predefine a threshold and trigger migration only after the shadow table's capacity exceeds this threshold.
- **Predictive:** Hermes can employ a predictive algorithm to estimate the size of the shadow tables and pre-emptively trigger migration if the predicted size indicates an overflow of the shadow table.

Hermes's current design opts for the predictive approach because it frees the programmer from the burden of rewriting the application while simultaneously ensuring that Hermes enforces its performance guarantees. As we will show in Section 8, the predictive approach is a better fit for Hermes than the Simple-Threshold because it allows Hermes to dynamically adjust to workload demands without incurring huge overheads.

The prediction algorithm takes as input a time series of rule arrival rates and creates as output future rule arrival rates. We explore a number of predictive algorithms ranging from Exponentially Weighted Moving Average (EWMA) [46] and Cubic Spline [34] to Autoregressive Moving Average (ARMA) [63].

In our empirical evaluation of these prediction algorithms, we observed prediction error due to dynamic and drastic changes in the workloads. We adjust to errors in the prediction algorithm by augmenting our prediction framework with control theoretic mechanisms for counteracting prediction errors. Specifically, we focus on the following promising approaches:

- **Slack:** To compensate for predictive errors, we inflate the predicted value by a constant factor, e.g., a slack of 40% (inflates the prediction by 40%). Given a predicted value of 1000 rules, a slack value of 40% would result in a predicted value of 1400.
- **Deadzone:** Alternatively, we can inflate the predicted shadow size by a constant value, e.g., 100 rules. Thus, given a predicted value of 1000 rules, applying deadzone would result in a predicted value of 1100.

Through extensive evaluations with large-scale production workloads and synthetic traces (Section 8), we found Cubic Spline and Slack to be the most effective.

5.2 How to migrate rules?

The Rule Manager performs migration in four steps (Figure 7). First, rules from the shadow table and the main table are copied into the Rule Manager. The Rule Manager, then, optimizes the migration process using TCAM optimization algorithms [43, 62] — these algorithms speed up the migration by rewriting the rules to optimize and minimize the number of rules. This reduction in the number of rules translates into a reduction in the migration time.

[4] Either the rule or its partitions can exist but never both.

[5] For example, the rules that will cause numerous partitions (e.g., 0.0.0.0/0 with the lowest priority which will overlap with all rules in the main Table), Hermes directly inserts them into the main Table.

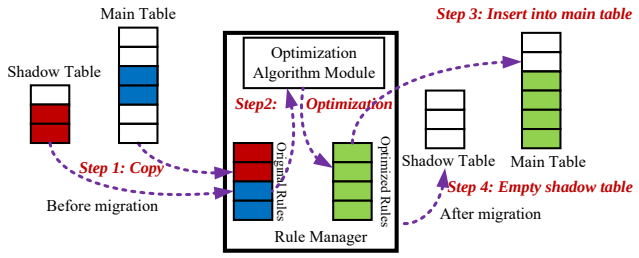


Figure 7: Rule Manager's Migration Workflow.

Third, the Rule Manager writes the optimized rules into the main table. Finally, the shadow table is emptied.

The migration process is designed to ensure that the Rule Manager maintains the following properties:

Correctness During Migration Consistency: During migration, Hermes explicitly does two things to maintain consistency in the switch.

First, the Rule Manager does not empty the shadow (step 4) until after migration is complete (step 3). This ensures that at any point in time there is at least one rule to process and service packets. While there may be two identical rules in both tables: one in the shadow table and one in the regular table. Fortunately, the default behavior for Hermes is to stop matching after the packet matches a rule in the shadow table.

Second, during step 3 when "optimized" rules are inserted into the main TCAM table, this insertion of rules can lead to correctness violations. For example, in a naive approach Hermes can delete the old rules before inserting the optimized rules, this ordering of events can create violations because the add and delete operations are not atomic [36, 42]. Without atomicity, a packet may arrive in the transient period between the deletion and insertion. One method to provide atomicity is to pause packet processing, stall the switch pipeline, during migration time; however, this impacts the data plane by slowing down data plane processing throughput and potentially adding some jitter in processing latencies.

In view of this, Hermes uses an incremental update algorithm that provides atomicity at the cost of a few additional rule entries: (i) for each "optimized" rule r, Hermes gets a list of rules in the main table, O that o_i overlaps with r; (ii) Hermes, then, changes the priority of r to be one priority higher than all rules in O; (iii) Hermes inserts r into the main table and subsequently deletes $o_i \in O$, this ensures that the rule either matches r or one of the rules in O;

Performance Guarantees under Adversarial Workloads: Provided the shadow table is emptied and a rule is never inserted when the shadow table is full, Hermes will be able to provide performance guarantees.

A challenge arises when rules are inserted faster than Hermes can empty out the shadow table. For example, if a batch of rules is inserted and the batch size is larger than the shadow table, then Hermes will be unable to provide guarantees for a subset of the rules. To address this issue, the Gate Keeper includes a rate limiter which performs admission control. Because the shadow table is fixed in size and capacity, Hermes provides specific performance guarantees as long as the application's insertion rate is below a predefined limit – a threshold that gives Hermes sufficient time to optimize and migrate rules. (Section 7)

The maximum insertion arrival rate of control plane actions, λ, that Hermes can support is a function of the size of the shadow table, S_{ST} (number of rules the shadow can support) and the speed with which rules can be migrated out of the shadow table, t_m, (rules per second).

$$\lambda = \frac{S_{ST}}{t_m} \tag{1}$$

Specifically, if rules are inserted faster than they can be migrated out, then the shadow table will become full and the Gate Keeper will be forced to wait for the shadow to be emptied or forced to use the main table. Both solutions may lead to performance violations.

To account for the fact that rules in the shadow table may be partitioned, equation 1 must be adjusted to included the expected number of partitions for each rule, r_p. Including the expected number of partitions effectively reduces the expected maximum supported insertion arrival rate (equation 2).

$$\lambda = \frac{S_{ST}}{r_p * t_m} \tag{2}$$

6 PRACTICALITY: IMPLEMENTATION FEASIBILITY

Switch hardware places a number of constraints on the design and implementation of Hermes. In this section, we discuss the design choices that enable Hermes to account for the practical constraints of the current and emerging switch hardware.

We have not implemented Hermes in hardware; however, our discussions with Broadcom engineerings as well as switch vendors indicate that Hermes can be implemented in the current line of switches using interfaces readily available in the Broadcom SDK. Unfortunately, SDK access is required because existing interfaces to switches, e.g., OFDPA [7] and OpenNSL [16], do not provide control over the configuration of TCAM slices. We are in the process of acquiring an NDA to implement Hermes on white box switches.

Carving TCAM into Slices: Modern and traditional commercial SDN switches provide mechanisms for partitioning TCAM tables, called TCAM carving or TCAM slicing [3, 13, 15]. For example, Cisco [12, 20] subdivides TCAM into 8 predefined slices and provides operators with configuration commands for specifying the number of entries in each slice. The Broadcom SDK maps each slice to a logical group and allows the switch to assign priorities to the different groups and to target insertion/deletion/modification operations to a specific group. In these devices, the hardware performs parallel looks across all slices (groups) in a TCAM table with each slice returning at most one match [2, 12]. The TCAM resolves conflicts across different slices using pre-configured priorities.

Implementing Hermes with Slices: To support Hermes, we can carve the TCAM into two slices or configure two slices depending on the switch. Both slices are configured with identical keys, and the shadow slice is configured to be significantly smaller than the main slice; the size of the shadow slice is a function of the configured performance guarantees.

Supporting Multiple TCAM Tables: Modern switches support multiple TCAM tables, while Hermes is designed for a switch with one table. Hermes addresses this evolution by independently carving each TCAM table to support a shadow and a main table. This independent decomposition of the different tables enables Hermes to provide different guarantees for different tables: a feature that may be particularly attractive when the different tables are used for radically different functionality [44, 61]. To preserve the semantics of the original pipeline, Hermes configures each main table to exhibit the default "miss" behavior of the original table – either go to the next table, send the packet to the controller, or drop the packet. Note, Hermes still configures the shadow tables to use the "goto the next table (main table)" behavior during a TCAM "miss".

7 NOVEL ABSTRACTIONS

Hermes exposes a simple interface that allows the controller or network operator to configure performance guarantees at the granularity of individual switches. The core function, *CreateTCAMQoS()*, takes as input the switch id of the switch to be configured, the desired performance guarantees, and a predicate defining the set of rules that should get these guarantees. This function returns the max burst rate (using Equation 2) that Hermes will support – this max burst rate is used by the Gate Keeper to perform admission control (Section 3). This function also returns a file descriptor that can be later used to delete (*DeleteQoS()*) or directly modify the shadow table (*ModQoSConfig()* or *ModQoSMatch()*). Finally, we include a method, *QoSOverheads()*, that aids the operators in systematically exploring the trade-offs between performance and TCAM overheads. Our design of the interface is informed by the hardware constraints that impact TCAM reconfiguration and slicing.

```
int CreateTCAMQoS ( SwitchID, perf-guarantee, match-predicate );
boolean DeleteQoS(ShadowID)
boolean ModQoSConfig(ShadowID, perf-guarantee)
boolean ModQoSMatch(ShadowID, match-predicate)
double QoSOverheads(SwitchID, perf-guarantee, match-predicate)
```

8 EVALUATION

Our evaluation of Hermes is driven by the following questions: What are the application level benefits of using Hermes? (§8.2) How does Hermes compare with state-of-the-art techniques? (§8.3) Do traditional control planes, e.g. BGP, benefits from employing Hermes? (§8.4) How effective are Hermes' smart and complex migration algorithms? (§8.5) How sensitive is the performance of Hermes to various system parameters? (§8.6) What are the overheads of employing Hermes? (§8.7)

8.1 Experiment Setup

First, we provide a brief overview of Varys, our network simulator (§ 8.1.1), the metrics (§ 8.1.2) and workloads (§ 8.1.3) used in our evaluations.

8.1.1 Network Simulator. We developed a flow level network simulator in Python in 1738 lines of code. Here we describe the key aspects of this simulator.

Switch Performance Model: We modeled control plane action latency by incorporating existing empirical models of switch TCAM behavior based. Specifically, for each switch on the path of a flow, the simulator uses the distributions from recent measurements [38, 57] to determine the insertion latencies. Our current simulator is designed to model switches whose TCAM behavior (specifically control plane actions) is a function of two factors: (1) the occupancy of the flow table, i.e., the current number of rules in the flow table, and (2) the properties of the rule being inserted and the rules before it. This TCAM behavior covers all dominant TCAM designs. Our simulator includes models for three common commodity switches [42]: Dell PowerConnect 8132F, HP 5406zl, and Pica8 P-3290. Our simulator can be easily extended to model other switches by incorporating the empirical models describing their TCAM patterns. Unless specified, our experiments are run across all three switch models.

SDN Application (SDNApp): In our simulation, we evaluated a proactive traffic engineering SDNApp [33] that periodically reconfigures the network by using control plane actions to move congested flows away from congested links unto links with available capacity. This SDNApp is impacted by slow control plane actions because they extend the period of time that a flow remains on the congestion path and thus inflate the flow's completion time and ultimately the job's completion time.

8.1.2 Metrics. The evaluation uses three major metrics: rule installation latency (latency), this metric captures the amount of time it takes a switch to install a given rule into TCAM. Job completion time (JCT), a metric, specific to big data workloads, that measures the time between the beginning of the job's first flow and the end of the job's last flow. Flow completion time (FCT), a metric which captures the duration between when the first packet of a flow is sent and when the last packet is received. Finally, the rule installation time (RIT) which captures the time to install rules in a switch.

8.1.3 Workloads and Topologies. We evaluated Hermes's performance using six distinct and representative datasets: *Facebook*, *Abilene*, *Geant*, *BGP-Updates*, *Synthetic-Traces*, and *Quest*. Below we describe key properties of these datasets.

Facebook [29]: This workload captures Facebook's large-scale Map Reduce deployment consisting of 24402 Map Reduce jobs run over 1 day on a 600-machine cluster. This cluster uses a large clos-style data center topology, which we emulate using a K=16 fat tree[21] with 1024 servers and 40 Gbps link speeds.

Abilene [1, 58]: This dataset includes the topology and traffic matrix from Internet2, a backbone ISP, from March 1st to September 11th, 2004. The traffic matrix captures traffic between the ISP's ingress and egress nodes. To simulate this workload, we route traffic over the topology using a wide-area traffic engineering technique [60]. We generate individual flows from the coarse-grained traffic matrix by assuming flow inter-arrivals follows a Poisson process and that flow sizes are partitioned evenly according to the total data given in the traffic matrices.

Synthetic workloads: To further understand the benefits of Hermes, we generated synthetic traffic matrices over two representative ISP topologies from the Internet Zoo archive. Specifically, a European research network (Geant [10]) and the Quest topology [19]. The traffic matrices are generated using the tomo-gravity

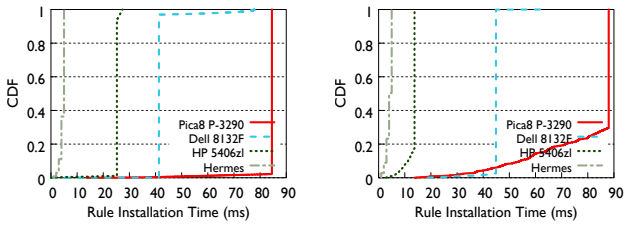
model [65]. The flows and network routes are generated in a similar matter to the Abilene model above.

MicroBench Traces: For microbenchmarks, we generated a stream of rule insertions in a systematic manner, varying the following three dimensions to understand the performance of Hermes: the arrival rate (to understand the impact of bursts), overlap rate (to understand the impact of partitioning), and priorities (to understand the impact of TCAM moving/rearrangement). The overlap rate is defined as the number of rules currently in the main and shadow tables that the new rule overlaps with. A new rule with 100% overlap rate means that it overlaps with all existing rules (e.g., with wildcard *). For these traces, we simulated a simple topology with just one switch to help us focus on the interactions between these dimensions, switch TCAM behavior, and Hermes.

BGPTrace: To understand the impact of Hermes on traditional networks and to evaluate realistic expectations of our partitioning algorithm. We evaluate BGP updates collected by BGPStream [5] from four representative high traffic routers: Equinix in Chicago, TELXATL in Atlanta, NWAX in Portland, and the University of Oregon. We preprocessed the BGPStream data by converting the BGP updates into Forwarding Information Base (FIB) rules that get inserted into TCAM. This preprocessing exposes only the FIB rules and not the RIB rules because many RIB updates do not percolate down to the FIB and it is the FIB rules that are installed into the TCAM.

8.2 Performance Benefits of Hermes

We begin by exploring the low-level performance benefits of Hermes and then analyzing how these low-level benefits impact higher layer application metrics. In these experiments, we focus on the following four workloads: Geant, Facebook, Abilene, Quest but only present results from the two representative workloads: Facebook and Geant.



(a) Facebook (b) Geant

Figure 8: Rule Installation Time.

Rule Level Benefits of Hermes: We begin, in Figure 8, by analyzing the rule installation latencies (RITs) for Hermes relative to traditional switches. In Figure 8, we observe that Hermes improves the median rule installation time by 86%, 94% and 80% across all switches. Moreover, we observe minor variations in the RIT provided by Hermes. These minor variations exist because while Hermes provides an upper bound on performance, Hermes may, in fact, perform better than this upper bound and thus installation time may be lower than expected.

Application Level Benefits of Hermes: Next, we explore higher-level metrics that are predictive of end-user and application perceived performance. Specifically, we explore the flow completion time (FCT) and the job completion time (JCT).

At a glance, we observe that as we move to higher layer metrics, the benefits of Hermes are less pronounced but pronounced nonetheless. At the FCT level (in Figure 9), we observe that with the Facebook trace Hermes improves the median flow completion time by up to 48%, 80% and 43% over the 8132F, the P-3290, and the 5406zl switches respectively, Moreover, we observe that at the JCT level (in Figure 1), Hermes improves the median by up to 38%, 42% and 31% respectively.

The application layer benefits (FCT and JCT) of Hermes are lower than RIT level benefits because the application layer performance is a function of many factors and resources in addition to the network. Specifically, the JCT is a function of rule installation times, flow transfer times and compute times whereas the FCT is a function of rule installation times and flow transfer times. To illustrate the importance of Hermes even for such application layer metrics, in Figure 9(b), we focus on short flows which allows us to analyze the impact of Hermes on flows where the impact of flow transfer times and computation times are minimal. From this figure, we observe that Hermes provides significant benefits with a 95-percentile improvement of around 80%, which is similar to the RIT level improvements.

8.3 Benefits of Hermes over Related Works

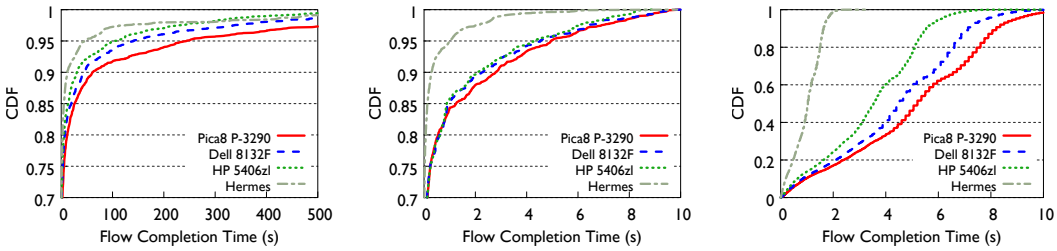
Next, we compare Hermes against the most closely related and impressive approaches, Tango [43] and ESPRES [51], for improving the latency of TCAM insertion times. Unlike Hermes which makes fundamental changes to the TCAM's behavior, Tango, and ESPRES improve performance by changing the ordering and structure of the rules being inserted.

We present our results in Figure 10. Our experiments show that all three approaches significantly improve the rule installation time over a traditional switch. However, from (Figure 10), we observe that the performance of Tango and ESPRES varies wildly when compared with Hermes. Additionally, Hermes significantly outperforms Tango and ESPRES by more than 50% in the median case, a benefit of fundamentally changing the behavior of the TCAM. Tango performs similarly to ESPRES and outperforms ESPRES at the tail because while ESPRES changes rule ordering, Tango changes both rule ordering and rewrites the rules – this additional degree of freedom helps at the tail.

To illustrate the rationale behind Hermes's performance, in Figure 11 we present a time series demonstrating the rule installation times for the first 1000 rules in our experiment.

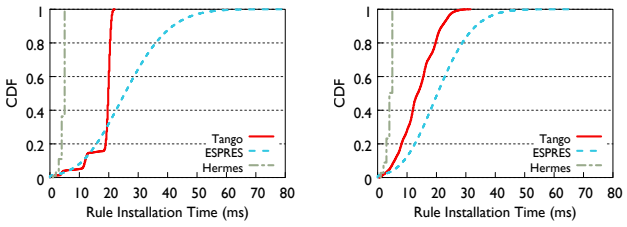
An initial observation is that across all algorithms, the insertion times grow slowly as more rules are added because rule shifting is performed to maintain correctness. However, after around rule #200, we start to notice key differences between the algorithms. At this point, the benefits of Hermes become apparent.

Figure 11(a), shows that initially Tango and ESPRES behave in a similar manner but then diverge shortly after rule #500 with ESPRES' performance growing worst over time. Further analysis shows, that Tango and ESPRES initially perform similarly because



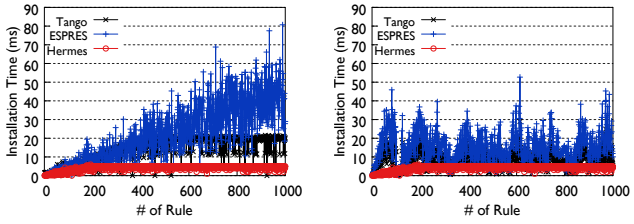
(a) Facebook (All Jobs) (b) Facebook (Short Jobs) (c) Geant

Figure 9: Flow Completion Time.



(a) Facebook (b) Geant

Figure 10: Comparison of Rule Installation Time.



(a) Facebook (b) Geant

Figure 11: Time Series of Rule Installation Time.

they are initially employing the same optimization: reordering rules. However, over time Tango also starts to take advance of properties of IP allocation and symmetry in the data center to aggregate and reduce the rules being inserted. By employing this additional optimization, Tango is able to outperform ESPRES. The key distinction between the behavior of Tango and ESPRES on Facebook compared with the Geant trace is due to the fundamental differences in the structure and IP allocation of data centers and ISP networks (e.g. Geant).

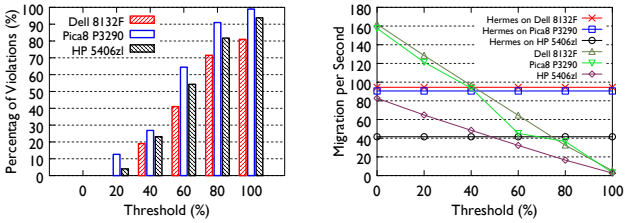
On the contrary, Hermes can always provide guarantees in both situations, this property arises because Hermes makes fundamental changes at the hardware level rather than changing the rules being inserted into the hardware.

8.4 Traditional Networks and Hermes

Here, we evaluate the effectiveness of Hermes (with a 5ms guarantee) within a traditional router running BGP and processing the BGP-updates discussed in Section 8.1. These experiments enable us to explore the sensitivity of Hermes's core algorithms to the change in workload from SDN to BGP. Due to space constraints, we omit the figures and summarize the results of these experiments. We observe that the algorithms behave similarly with BGP as they did with the SDNApp — with Cubic+Slack providing the best performance and with Hermes requiring high slack inflation (over 80%) to ensure that there are no performance violations. Similarly, we compare the performance of Hermes in terms of rule installations times and observe that the benefits of employing Hermes are significant and nontrivial.

8.5 Benefits of Smart Migration Techniques

Next, we evaluate the broader impact of having smart migration algorithms by comparing Hermes against a version of Hermes with a very simple migration algorithm that uses a threshold to determine migration. This simple version of Hermes is called *Hermes-SIMPLE*. The insertion rate is set to 1000 updates/s, and the overlap rate is 100%. In Figure 12, we examine the performance and overheads of Hermes-SIMPLE under varying threshold values. We observe in Figure 12 (a) that Hermes-SIMPLE has no violations when the threshold is 0% meaning that migration is constantly happening in the background. In Figure 12 (b), we compared Hermes-SIMPLE with regular Hermes, for a fair comparison, the slack value used in Hermes is set to 100% since it is the minimal value for zero violation. We observe that the end result of constantly migrating is that Hermes-SIMPLE incurs double the overheads of Hermes. Essentially, the smart algorithms introduced into Hermes provide strong guarantees at a lower cost (migration overheads) than a naive approach.



(a) Percentage of Violations (b) Migration Frequency

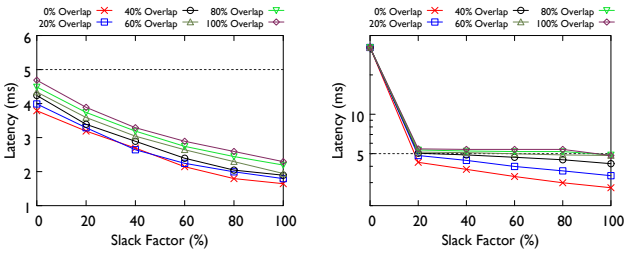
Figure 12: Hermes-SIMPLE Performance under Different Threshold Values.

8.6 Sensitivity Analysis

In this section, we analyze the sensitivity of Hermes to our smart migration techniques (prediction and correction algorithms) and the parameters for these techniques and algorithms.

Sensitivity to Prediction Algorithms: For this analysis, we used the *MicroBench Traces* with the simplified topology. We evaluate Hermes over the various predictive (EWMA, Cubic Spline, and ARMA) and corrective (Slack, Deadzone) algorithms presented in Section 5.1.

We observed that Cubic Spline provided the lowest prediction error, especially when combined with Slack. We observed that the combination of Cubic Spline and Slack reduced rule installation time by 80% – 94% over existing alternatives (EWMA+Slack, EWMA+Deadzone, Cubic Spline+Deadzone).



(a) 200 updates/s (b) 1000 updates/s

Figure 13: Rule Insertion Latency.

Sensitivity to Parameters: Next, we dig deeper to understand how our choice of parameters impacts the performance of Hermes. In Figure 13, we present the result of analyzing the impact of different slack values on Hermes across two distinctly different rule update rates (200 (a) and 1000 (b)) on a Dell 8132F switch. [6]

The higher update rate (Figure 13 (b)) creates more partitions and, in turn, requires a more aggressive slack value. Whereas for lower update rates (Figure 13 (a)), the slack does not impact Hermes's ability to make guarantees but it does help Hermes provide better performance.

A slack of 100% is required to appropriately tackle the high insertion rates (1000 flows per second) and for lower insertion rates (200 flows per second) less drastic slack values are required. This experiment demonstrates the need to empirically adjust Hermes to the properties of the network. As part of future work, we will explore learning techniques to enable Hermes to automatically tune itself.

Motivated by the above results, Hermes is by default configured to Cubic Spline with a slack inflation of 100% unless otherwise specified.

8.7 Overhead

Next, we analyze the overheads of employing Hermes on the switch's ASIC, CPU, and memory.

ASIC Storage: The overheads of employing Hermes are directly proportional to the performance guarantees required and the size of the shadow table required to satisfy these guarantees. In Figure 14, we present the overheads for different performance guarantees. The figure demonstrates that while the overheads vary across switches, the overheads are small and acceptable.

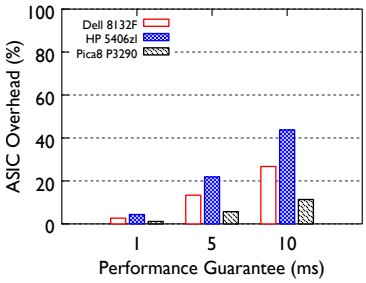
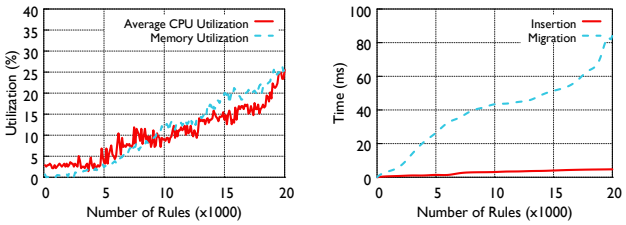


Figure 14: ASIC Overhead Percentage.

CPU and Memory: Running Hermes's insertion and migration algorithms requires CPU/Memory resources and introduces additional latencies. Next, in Figure 15, we present the results of running our algorithms on a physical switch (Edge-Core AS5712). In these experiments, we varied the arrival rates of the rules inserted and migrated between 100 and 20000 rules per second. For the experiment, we used the BGPTrace data with the simple topology. The results show that the CPU and memory utilization grows linearly with the number of rules being processed by the algorithm demonstrating the scalability of Hermes core algorithms. We further expect these overheads to be reduced once the algorithms are implemented in more effective and effective languages, e.g., C.

Algorithm Runtimes: We conclude by evaluating the runtimes of Hermes's algorithms. In Figure 15 (b), we observe that runtimes for the insertion algorithms are relatively constant and grow slowly over time. This is extremely beneficial as large runtimes would directly impact Hermes's ability to make guarantees. Unlike the insertion algorithm, we observe that the migration algorithm have a cubic growth pattern. Fortunately, the migration happens within the background and doesn't impact Hermes's ability to enforce guarantees. Moreover, recall from Section 8.6 that Hermes explicitly includes an algorithm to ensure these migration overheads do not impact performance guarantees.



(a) CPU/Memory Utilization (b) Processing Time

Figure 15: Hermes Overhead.

[6] We note that these experiments were conducted across all three switches and the results were qualitatively similar.

8.8 Takeaway

To summarize, Hermes significantly improves the performance of the network and the applications running atop the network. The applications which require frequent modifications will yield significantly more benefits with Hermes. This is only natural as Hermes is only involved when the network is being reconfigured with TCAM control plane actions.

9 RELATED WORK

Modeling Switch Performance: Prior works [32, 37, 43, 55] have conducted empirical studies on the factors that impact control plane action latency. These studies motivate our work and inspire the design of Hermes.

Improving TCAM Performance: Traditionally, work [31, 53, 66] to improve TCAM performance has focused on improving TCAM lookup latencies. In this paper, we attack an orthogonal problem, TCAM modifications, which has recently become equally important because of the increased frequency of TCAM modifications demanded by SDN control applications.

The most closely related work, ShadowSwitch [26], introduces a software-based table to control TCAM insertion times whereas Hermes uses a hardware-based table. The use of a hardware-based table enables Hermes to explore an alternate point in the design space with a distinctly different set of prediction and migration algorithms and system design.

To improve TCAM performance, existing approaches either reorder rules [41, 43, 45, 52] or change the TCAM insertion algorithms [62]. Even though these approaches reduce control plane action latency, they do not provide any guarantees or assurances. Hermes addresses exactly this: Hermes provides performance guarantees over control plane action latencies. Additionally, our evaluations (§ 8) show that Hermes outperforms state-of-the-art techniques [43, 51] under a variety of workloads.

TCAM Management: While Hermes divides a logical TCAM into two physical TCAM tables to provide performance guarantees, others have looked into similar partitioning to provide orthogonal properties, such as consistent updates [36]. While Hermes focuses on improving TCAM performance others [49] have explored methods for improving switch DRAM performance by using SRAM as a cache.

Log-Structured File Systems: Hermes is in principle similar to log-structured work in the file system community, most notably the log-structured file system (LSFS) [54] and Log-structured merge tree [50], which improves the performance of disk IO operations by writing to faster but smaller disk media and periodically moving the data to larger but slower media once the data gets too large. Due to the fundamental differences between disks and networks, the design of Hermes's insertion and the merge algorithms which leverage unique properties of the network.

10 DISCUSSION

Other Control Plane Actions: Hermes's current design focuses on a subset of SDN control plane actions, namely, rule insertion, modification, and deletion. As part of ongoing work, we are extending Hermes to make performance guarantees over other control plane actions by using a combination of techniques ranging from offloading switch's CPU functionality to developing resource allocation algorithms that rate-limit different events.

Emerging Programmable Data Planes (P4): Our current prototype and design are based on properties of modern merchant silicon and ASICs. Yet, Hermes should be applicable to the emerging generation of programmable data planes, e.g., P4 and RMT chips [27]. We believe that the core design of Hermes is applicable to these emerging hardware platforms because these platforms use TCAM-based flow tables for their flexible matching and while they may use different TCAM hardware components, Hermes addresses a property of TCAM that is invariant to underlying hardware components. Moreover, we believe that P4 will enable the adoption of Hermes because P4 provides direct control over the hardware which will enable the development and deployment of Hermes without requiring access to proprietary SDKs.

11 CONCLUSION

SDN control programs, e.g. traffic engineering and service chaining, require frequent modification of a switch's TCAM using control plane actions. Moreover, many of these SDN control programs require their actions to be completed in a timely manner. Unfortunately, modern SDN switches do not provide concrete performance guarantees for control plane actions – instead, the switches provide a best-effort service. To support many emerging applications and scenarios, we must redesign switch software and algorithms to explicitly support frequent control plane actions.

In this paper, we propose Hermes, a practical and immediately applicable system which provides strict performance guarantees for control plane actions by intelligently partitioning and managing TCAM flow tables. Hermes provides performance guarantees by trading-off a modest amount of TCAM space for enhanced TCAM performance. Specifically, Hermes keeps the occupancy of a small number of flow tables fixed to below a threshold to enable bounded insertion times. We evaluated Hermes using large-scale simulations, our evaluations show that with less than 5% overheads, Hermes provides 5 ms insertion guarantees that translates into an improvement of application level metrics by up to 80%.

ACKNOWLEDGMENTS

We thank Roberto Bifulco, Keqiang He, Robert Soule, Brent Stephens, our shepherd Michael Schapira, and the anonymous CoNEXT reviewers for their invaluable comments. This work is partially supported by NSF grant CNS-1409426, National Basic Research Program of China 2013CB329103, NSFC fund 61671130, 61271165.

REFERENCES

[1] The Abilene topology and traffic matrices. http://epubs.surrey.ac.uk/807659/.
[2] ACL Solutions Guide. http://extrcdn.extremenetworks.com/wp-content/uploads/2014/10/ACL_Solutions_Guide.pdf.
[3] AS5712. http://www.edge-core.com/productsInfo.php?cls=1&cls2=8&cls3=44&id=15.
[4] Barefoot. https://www.barefootnetworks.com/technology/#tofino.
[5] BGPStream. https://bgpstream.caida.org/.
[6] Broadcom. Personal communications, Dec 2016 - Jan 2017.
[7] Broadcom SDN Solutions OF-DPA. https://www.ietf.org/proceedings/90/slides/slides-90-sdnrg-3.pdf.
[8] CAM (Content Addressable Memory) VS TCAM (Ternary Content Addressable Memory). http://goo.gl/EpV4gr.
[9] Facebook Production HDFS. http://goo.gl/myVz6b.
[10] GEANT topology map. http://goo.gl/mLFq5P.

[11] Migration Use Cases and Methods. https://www.opennetworking.org/images/stories/downloads/sdn-resources/use-cases/Migration-WG-Use-Cases.pdf.
[12] Nexus 9000 TCAM Carving. http://goo.gl/wXC0KY.
[13] NOVISWITCH. http://noviflow.com/products/noviswitch/.
[14] Open Compute Project. http://opencompute.org/.
[15] OpenFlow Switch Specification v1.3. http://goo.gl/pCF6Hf.
[16] OpenNSL. https://github.com/Broadcom-Switch/OpenNSL.
[17] The P4 Language Specification v1.0.3. http://goo.gl/9NN44P.
[18] SDN System Performance. http://www.pica8.com/pica8-deep-dive/sdn-system-performance/.
[19] SOL: SDN optimization layer. A framework for writing network optimization problems more easily. http://goo.gl/k1W4Wg.
[20] Understanding and Configuring Switching Database Manager on Catalyst 3750 Series Switches. http://goo.gl/nLziyq.
[21] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of ACM SIGCOMM 2008*.
[22] B. Anwer, T. Benson, N. Feamster, and D. Levin. Programming slick network functions. In *Proceedings of ACM HotSDN 2015*.
[23] R. Baheti and H. Gill. 2011. Cyber-physical systems. *The impact of control technology* 12 (2011), 161–166.
[24] G. Bianchi, M. Bonola, A. Capone, and C. Cascone. 2014. OpenState: programming platform-independent stateful openflow applications inside the switch. *ACM SIGCOMM Computer Communication Review* 44, 2 (2014), 44–51.
[25] G. Bianchi, M. Bonola, S. Pontarelli, D. Sanvito, A. Capone, and C. Cascone. 2016. Open Packet Processor: a programmable architecture for wire speed platform-independent stateful in-network processing. *arXiv preprint arXiv:1605.01977* (2016).
[26] R. Bifulco and A. Matsiuk. 2015. Towards scalable sdn switches: Enabling faster flow table entries installation. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 343–344.
[27] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. In *Proceedings of ACM SIGCOMM 2013*.
[28] H. Chen and T. Benson. The Case for Making Tight Control Plane Latency Guarantees in SDN Switches. In *Proceedings of ACM SOSR 2017*.
[29] M. Chowdhury, S. Kandula, and I. Stoica. Leveraging endpoint flexibility in data-intensive clusters. In *Proceedings of ACM SIGCOMM 2013*.
[30] M. Chowdhury, Y. Zhong, and I. Stoica. Efficient coflow scheduling with Varys. In *Proceedings of ACM SIGCOMM 2014*.
[31] P. T. Congdon, P. Mohapatra, M. Farrens, and V. Akella. 2014. Simultaneously reducing latency and power consumption in openflow switches. *IEEE/ACM Transactions on Networking (TON)* 22, 3 (2014), 1007–1020.
[32] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. DevoFlow: scaling flow management for high-performance networks. In *Proceedings of ACM SIGCOMM 2011*.
[33] A. Das, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and C. Yu. Transparent and Flexible Network Management for Big Data Processing in the Cloud. In *Proceedings of USENIX HotCloud 2013*.
[34] C. De Boor, C. De Boor, E.-U. Mathématicien, C. De Boor, and C. De Boor. 1978. *A practical guide to splines*. Vol. 27. Springer-Verlag New York.
[35] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul. FlowTags: enforcing network-wide policies in the presence of dynamic middlebox actions. In *Proceedings of ACM HotSDN 2013*.
[36] J. H. Han, P. Mundkur, C. Rotsos, G. Antichi, N. H. Dave, A. W. Moore, and P. G. Neumann. Blueswitch: enabling provably consistent configuration of network switches. In *Proceedings of ACM ANCS 2015*.
[37] K. He, J. Khalid, S. Das, A. Akella, E. L. Li, and M. Thottan. 2014. Mazu: Taming latency in software defined networks. *University of Wisconsin-Madison Technical Report* (2014).
[38] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, and M. Thottan. Measuring control plane latency in SDN-enabled switches. In *Proceedings of ACM SOSR 2015*.
[39] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving High Utilization with Software-driven WAN. In *Proceedings of the ACM SIGCOMM 2013*.
[40] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, and others. B4: Experience with a globally-deployed software defined WAN. In *Proceedings of ACM SIGCOMM 2013*.
[41] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer. Dynamic scheduling of network updates. In *Proceedings of ACM SIGCOMM 2014*.
[42] M. Kuźniar, P. Perešíni, and D. Kostić. What you need to know about sdn flow tables. In *Proceedings of PAM 2015*.
[43] A. Lazaris, D. Tahara, X. Huang, E. Li, A. Voellmy, Y. R. Yang, and M. Yu. Tango: Simplifying sdn control with automatic switch property inference, abstraction, and optimization. In *Proceedings of ACM CoNEXT 2014*.
[44] Y. Li, G. Yao, and J. Bi. Flowinsight: decoupling visibility from operability in SDN data plane. In *Proceedings of ACM SIGCOMM 2014*.
[45] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. Maltz. zUpdate: Updating data center networks with zero loss. In *Proceedings of ACM SIGCOMM 2013*.
[46] J. M. Lucas and M. S. Saccucci. 1990. Exponentially weighted moving average control schemes: properties and enhancements. *Technometrics* 32, 1 (1990), 1–12.
[47] M. Moshref, A. Bhargava, A. Gupta, M. Yu, and R. Govindan. Flow-level state transition as a new switch primitive for SDN. In *Proceedings of ACM HotSDN 2014*.
[48] M. Moshref, M. Yu, R. Govindan, and A. Vahdat. DREAM: dynamic resource allocation for software-defined measurement. In *Proceedings of ACM SIGCOMM 2014*.
[49] S. Narayana, A. Sivaraman, V. Nathan, M. Alizadeh, D. Walker, J. Rexford, V. Jeyakumar, and C. Kim. Hardware-Software Co-Design for Network Performance Measurement. In *Proceedings of ACM HotNets 2016*.
[50] P. OâĂŹNeil, E. Cheng, D. Gawlick, and E. O'Neil. 1996. The log-structured merge-tree (LSM-tree). *Acta Informatica* 33, 4 (1996), 351–385.
[51] P. Pereíni, M. Kuzniar, M. Canini, and D. Kostić. ESPRES: transparent SDN update scheduling. In *Proceedings of ACM HotSDN 2014*.
[52] P. Perešíni, M. Kuzniar, N. Vasić, M. Canini, and D. Kostiū. OF. CPP: Consistent packet processing for OpenFlow. In *Proceedings of ACM HotSDN 2013*.
[53] V. Ravikumar and R. N. Mahapatra. 2004. TCAM architecture for IP lookup using prefix properties. *IEEE Micro* 24, 2 (2004), 60–69.
[54] M. Rosenblum and J. K. Ousterhout. 1992. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems (TOCS)* 10, 1 (1992), 26–52.
[55] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore. OFLOPS: An open framework for OpenFlow switch evaluation. In *Proceedings of PAM 2012*.
[56] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, and S. Licking. Packet transactions: High-level programming for line-rate switches. In *Proceedings of ACM SIGCOMM 2016*.
[57] D. Tai, H. Dai, T. Zhang, and B. Liu. On Data Plane Latency and Pseudo-TCP Congestion in Software-Defined Networking. In *Proceedings of ACM/IEEE ANCS 2016*.
[58] P. Tune and M. Roughan. 2013. Internet traffic matrices: A primer. *Recent Advances in Networking* 1 (2013), 1–56.
[59] B. Vamanan, G. Voskuilen, and T. Vijaykumar. EffiCuts: optimizing packet classification for memory and throughput. In *Proceedings of ACM SIGCOMM 2010*.
[60] Y. Vanaubel, P. Mérindol, J.-J. Pansiot, and B. Donnet. MPLS under the microscope: Revealing actual transit path diversity. In *Proceedings of ACM IMC 2015*.
[61] A. Wang, Y. Guo, F. Hao, T. Lakshman, and S. Chen. Umon: Flexible and fine grained traffic monitoring in open vswitch. In *Proceedings of ACM CoNEXT 2015*.
[62] X. Wen, B. Yang, Y. Chen, L. E. Li, K. Bu, P. Zheng, Y. Yang, and C. Hu. RuleTris: Minimizing Rule Update Latency for TCAM-based SDN Switches. In *Proceedings of IEEE ICDCS 2016*.
[63] P. Whitle. 1951. *Hypothesis testing in time series analysis*. Vol. 4. Almqvist & Wiksells.
[64] M. Yu, L. Jose, and R. Miao. Software Defined Traffic Measurement with OpenSketch.. In *Proceedings of USENIX NSDI 2013*.
[65] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg. Fast accurate computation of large-scale IP traffic matrices from link loads. In *Proceedings of ACM SIGMETRICS 2003*.
[66] K. Zheng, C. Hu, H. Lu, and B. Liu. 2006. A TCAM-based distributed parallel IP lookup scheme and performance analysis. *IEEE/ACM Transactions on Networking (TON)* 14, 4 (2006), 863–875.