FlowShadow: Keeping Update Consistency in Software-based OpenFlow Switches

Yi Wang[†‡], Dongzhe Tai[‡], Ting Zhang[‡], Bin Liu[‡©]
[†]Huawei Future Network Theory Lab, Hong Kong
[‡]Tsinghua National Laboratory for Information Science and Technology,
Department of Computer Science and Technology, Tsinghua University

***Abstract*—The fast path, as the cache of exact-match rules in the slow path, is applied in software-based OpenFlow switches to improve the forwarding performance. A microflow in the fast path is the specification of its corresponding rules in the slow path, i.e., every field is explicit in a microflow. A rule can generate multiple microflows in the fast path, and a microflow can be generated from multiple rules since there are multiple flow tables in an OpenFlow switch. Due to the many-to-many mapping relationship between the microflows and the rules, the update consistency between the slow path and the fast path becomes a big challenge in software switches, e.g., Open vSwitch (OVS). In this paper, we propose a cache-based scheme (named *FlowShadow*) to achieve high update performance while keeping update consistency in OVS. In order to examine the reliability, validity, utility and scalability of FlowShadow, we implement FlowShadow on the OVS and conduct numerous experiments with different settings to measure the performance of FlowShadow. The experimental results demonstrate that FlowShadow achieves a lookup speed of 75 million packets per second on a commodity PC under the real backbone traces; the system with FlowShadow speeds up 3.4× times of the original OVS; and FlowShadow also shows high update performance and good scalability at different update speeds and with different numbers of flow tables.**

***keywords*—Software-Defined Networks, Update Consistency, OpenFlow, Open vSwitch, Action Table.**

I. Introduction

Software-Defined Networking (SDN) allows various network applications to process packets by deploying fine-grained rules on the flow tables of switches [1]. Rules, which fulfills the semantics of policies in controllers, identify flows[1] by matching multiple header fields. The finer the granularity of policies in the controller, the larger the number of rules in the switches. More and more applications are integrated into SDN networks to satisfy the increasing requirements (e.g., access control, forwarding, packet classification, and traffic managing). Multiple functions from these applications lead to numerous policies and fine-grained rules.

Updating rules in flow tables is complex and time-consuming. Often, a single rule change leads to multiple rules switched into the flow tables because rules are interdependent.

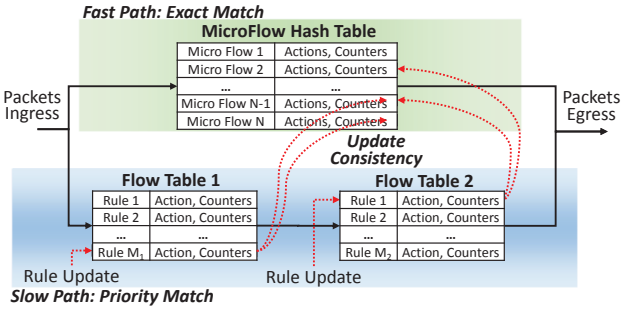


Fig. 1: The challenge of update consistency: one rule in the flow table (slow path) corresponds to multiple micro flows in the cache (fast path), and one microflow can have associated with multiple rules.

The lookup process is suspended while the flow tables being updated. Frequent rule updates cannot be avoided in current commodity SDN switches, which dramatically degrades the lookup performance and exacerbates the processing latency. Therefore, *it is imperative to find a solution that can keep packets being processed while the flow tables are updated.*

Caching the microflows is a potential solution. Based on the observation that network communications exhibit strong locality, Congdon *et al.* [2] use a prediction circuit before the flow tables to speedup flow classification of incoming packets. However, the prediction circuit cannot preserve the counters of rules and does not address the issue of statistics. Open vSwitch [3] (OVS), an open-source software implementation of an OpenFlow switch, implements two datapaths, the *fast path* and the *slow path*, to process the incoming packets. The fast path is the cache of exact-match rules, and the slow path is the original flow table. The first packet of each microflow undergoes the slow path to identify the highest-priority matching wildcard rule [4], and subsequent packets of the microflows in the cache are processed by the fast path. OVS's fast path stores the reference of rules in the microflows' associated data to preserve the rule counters and keep update consistency, but it only supports the single flow table (e.g., OpenFlow 1.0 [5]). Meanwhile, the lookup performance of the OVS' fast path is poor since each packet undergoing the fast path still needs to match the rule to get the final action.

Generally, applying cache to support uninterrupted forward-

©Corresponding author: Bin Liu, liub@tsinghua.edu.cn. This work is sponsored by Huawei Innovation Research Program (HIRP), NSFC (61373143, 61432009), the Specialized Research Fund for the Doctoral Program of Higher Education of China (20131019172), 863 project (2013AA013502), China Postdoctoral Science Foundation (No. 2015T80089), and Jiangsu Future Networks Innovation Institute: Prospective Research Project on Future Networks (No. BY2013095-1-03).

[1]In this paper, we use the terms: *flow* and *microflow* interchangeably.

ing while updating the flow tables, a scheme should overcome the following challenges:

1) *Update consistency*. When a rule is changed, the replica of this rule in the cache should be updated immediately to prevent inconsistency, which would cause the microflows of this rule to be handled in different actions. A rule is the *superset* of the microflows. As depicted in Figure 1, one rule in the flow table may correspond to multiple microflows in the cache; on the other side, one microflow in the cache can also have associated with multiple rules in the flow tables. Therefore how to fast update all the corresponding microflows of a modified rule in the cache is the key problem of achieving update consistency.
2) *High lookup speed*. The lookup performance of the cache directly determines the whole system's performance. Achieving high lookup performance is a basic but challenging requirement.
3) *Good scalability*. The number of flow tables are dynamic in the switches. The cache scheme should adapt to this variation and achieve stable performance regardless of the number of flow tables.

In this paper, we address the above challenges by applying cache in SDN switches. In doing so, we propose FlowShadow, a general solution based on cache, to keep working continuously with frequent rule updates. More specifically, our central contributions include:

1) The design of a new scheme for keeping update consistency between the multiple flow tables and the cache. We organize the actions of the microflows in a hash table (named *Action Table*), and leverage the state of each action (valid, or invalid) to indicate the states of the microflows' corresponding rules. Many microflows can share the same entry in the Action Table that is small enough to fit into the on-chip memory.
2) The design of a new data structure for caching microflows and their corresponding counters, actions and other associated data. The data structure has fast lookup speed and high cache replacement performance.
3) The verification implementation of FlowShadow based on the OVS (release version 2.1.2 [6]). To examine the reliability, validity, utility and scalability of FlowShadow, we modify the *fast path* of OVS' datapath to implement the cache mechanism and the update scheme of FlowShadow. Besides, we implement a module for supporting multiple flow tables by reusing the existing modules of the single flow table in OVS.
4) The experiments of FlowShadow in terms of cache hit rate, lookup speed, scalability on both campus traces and backbone traces are conducted. Experimental results demonstrate that FlowShadow achieves around 75 Mpps (Million packets per second) on a commodity server (24 physical threads) and the entire datapath of OVS with FlowShadow achieves 52 Mpps under the backbone traces; meanwhile FlowShadow has good scalability.

TABLE I: The details of the campus traces and the enterprise traces.

Trace Name	BW (Gbps)	Time	Average Utilization	Total MicroFlows
Campus-1	10	8:00∼8:05	39.09%	72,852,398
Campus-2	10	10:00∼10:05	49.71%	130,209,177
Campus-3	10	12:00∼12:05	51.50%	85,898,796
Campus-4	10	14:00∼14:05	53.35%	89,107,315
Campus-5	10	16:00∼16:05	53.93%	90,239,935
Enterprise-1	10	16:20∼16:25	23.20%	12,850,677
Enterprise-2	10	16:40∼16:45	23.86%	12,918,577
Enterprise-3	10	17:00∼17:05	29.58%	12,703,886
Enterprise-4	10	17:20∼17:25	25.36%	12,137,567
Enterprise-5	10	17:40∼17:45	21.42%	9,658,666
Enterprise-6	10	18:00∼18:05	25.31%	10,773,725

The rest of this paper is organized as follows. The motivation and the problem statement are introduced in Section II. In Section III, we describe the framework of FlowShadow, as well as the schemes for achieving fast update consistency. Then the extensive experimental results conducted on both the campus traces and the Internet backbone traces are demonstrated in Section IV. After reviewing the most related work in Section V, we conclude our work in Section VI.

II. Motivation and Problem Statement

A. Motivation

Network communications have strong locality, which inspires us to apply the cache schemes to speed up lookup process against flow tables. To begin with, we demonstrate the update requirements and the temporal localities of different types of traces: the campus traces and the enterprise traces. The campus traces, each trace lasting 5 minutes, are collected in 5 different time slots from a 10-Gbps interface card of an egress router in the campus network of Tsinghua University which contains more than 25,000 users. The enterprise traces, each trace also lasting 5 minutes, are collected in 6 different time slots from a 10-Gbps interface card of a core router in the network of Chinese Academy of Sciences which contains more than 3,000 users. Details of traces are listed in Table I.

1) The generating speed of new microflows: We use 5-tuple (Source IP, Destination IP, Source Port, Destination Port, and Protocol) to identify a microflow. The generating speed of new microflows in the real SDN switches can be very fast, since more fields are used to distinguish different microflows and the granularity of microflow is much finer than 5-tuple. The flow tables are empty in the beginning, any packet that cannot be found in the flow tables is identified as the first packet of a new microflow. Therefore, we consider the first 60 seconds as the warm-up time, and the rest of the traces are used to calculate the generating speeds of new microflows.

Figure 2 and Figure 3 illustrate the generating speeds of new microflows in the campus traces and the enterprise traces, respectively. In the campus traces, 200K $\sim$ 300K new microflows are generated per second. In the enterprise traces, the microflow generating speeds are around 50K per

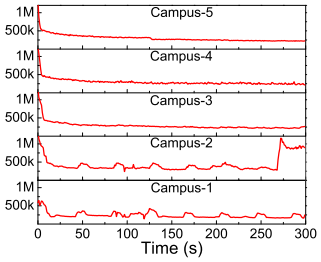


Fig. 2: The generating speed of new microflows in the campus traces.

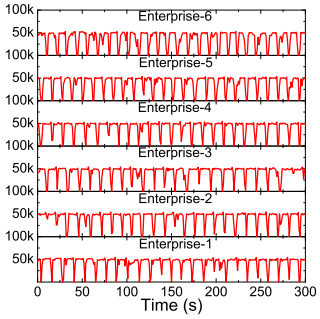
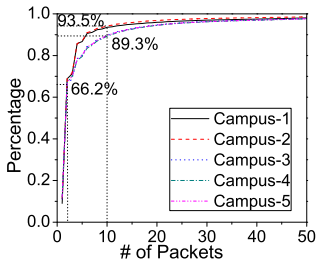
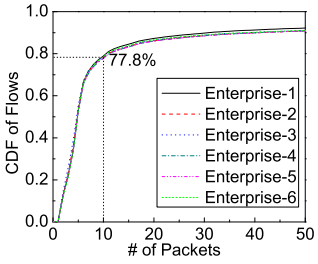


Fig. 3: The generating speed of new microflows in the enterprise traces.

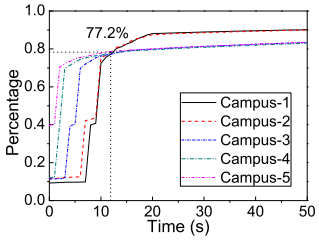


(a) The campus traces.

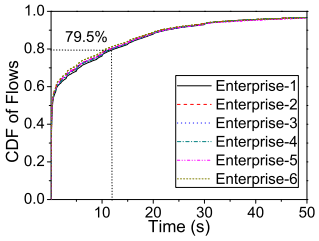


(b) The enterprise traces.

Fig. 4: The CDF of microflows in the campus and the enterprise traces, where the packet number of a microflow is less than #.

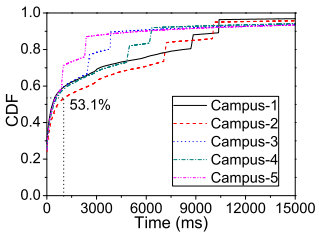


(a) The campus traces.

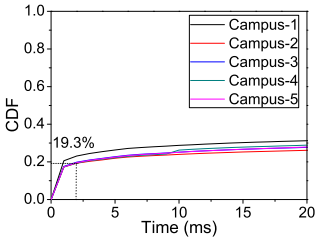


(b) The enterprise traces.

Fig. 5: The CDF of each microflow's duration in the campus and the enterprise traces.



(a) Macro View.



(b) Micro View.

Fig. 6: The CDF of intra-flow packet intervals in the campus traces.

second. In the most time, the microflow generating speed is stable. However, the bursts of generating new microflows are also possible. For example, in the Campus-2 trace, there are 279,233 new microflows in the 268-th second; then the number of new microflows grows to 557,918, 746,488, 873,550, and 1,154,460 in the next 4 seconds.

In summary, we can conclude that: 1) Compared to the enterprise network, the campus network generates more microflows; 2) If we deploy SDN switches in the campus and the enterprise networks, the flow tables should support the fast updates, especially have the ability to handle the update bursts.

2) The packet number of a microflow: The Cumulative Distribution Function (CDF) of microflows according to their packet numbers are illustrated in Figure 4(a) and Figure 4(b), respectively. As well as many results of the measurement-based studies [7], [8], [9], the microflow statistics of the campus and the enterprise traces also exhibit strong long-tail behaviors, i.e., the elephant and mice phenomenon. Most microflows (mice flows) have a small number of packets, while very few microflows (elephant flows) have a large number of packets. In the campus traces, more than 89.3% microflows have less than 10 packets; and in the enterprise traces, more than 77.8% microflows have less than 10 packets. Each microflow, whether it is an elephant flow or a mice flow, may trigger the controller to insert a new rule to the switch's physical flow table. Therefore, supporting a large number of updates caused by mice flows should be a basic and important function of the flow table.

3) The duration of a microflow: Besides the elephant and mice phenomenon, the other long-tail behavior of the microflows in the campus and the enterprise traces, is the long flows and the short flows. As many studies on the characteristics of the Internet flows [10], [11], the campus traces and the enterprise traces also exhibit that short flows dominate the traffics, which are demonstrated in Figure 5(a) and Figure 5(b), respectively. More than 77.2% microflows in the campus traces and 79.5% microflows in the enterprise traces live shorter than 12 seconds. The cache scheme should leverage this feature to improve the lookup speed by reducing the complexity of update operations.

4) The intra-flow packet intervals: The packets, belonging to the same microflow, will match the same rule or the same set of rules in the flow tables. The following packets of a microflow will be blocked, if their corresponding rules are unavailable. That means the lookup performance will be decreased, furthermore the packet forwarding performance will be degraded. Figure 6(a) shows that more than 53.1% intra-flow packet intervals are less than 1,000 ms. Specifically, as illustrated in Figure 6(b), more than 19.3% intra-flow packet intervals are less than 2 ms. Consequently, the insertions, modifications and deletions of microflows in the cache should be as soon as possible.

B. Problem Statement

The header space $\mathcal{H} = \{0,1\}^{\ell}$ is the logical space defined by the ℓ-bit packet header [12]. Here, the protocol-specific

semantics associated with header bits are ignored, and a packet header is considered as a flat sequence of 0s and 1s. For instance, the header space of the conventional 5-tuple flow definition is $\ell = 104 = 32 + 32 + 16 + 16 + 8$.

A packet header or microflow $\hbar$ is a point in the header space $\mathcal{H}$, and a rule $\mathcal{R}$ is a region in the header space $\mathcal{H}$. A rule in $\mathcal{H}$ is defined as a union of wildcard expressions, which are the basic building blocks used to define regions in $\mathcal{H}$. A wildcard expression is a sequence of ℓ bits where each bit can be either 0, 1 or x. The expression $\hbar_i \in \mathcal{R}_j$ means the point of packet header $\hbar_i$ is in the region of rule $\mathcal{R}_j$; in other words, the packet header $\hbar_i$ matches the rule $\mathcal{R}_j$.

An action $\mathcal{A}$ is an execution indicator, and a pair $(\mathcal{R}_j, \mathcal{A}_j)$ is the basic unit of a flow table. A flow table $\mathcal{T}$ is a set of these pairs: $\mathcal{T} = \{(\mathcal{R}_1, \mathcal{A}_1), (\mathcal{R}_2, \mathcal{A}_2), ..., (\mathcal{R}_m, \mathcal{A}_m)\}$. The Multiple Flow Table is the cascade of several single flow tables: $\mathcal{MT} =< \mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_n >$. Here, m is the number of rules in a flow table and n is the number of flow tables.

Looking up a packet header $\hbar_i$ against a flow table $\mathcal{T}_k$ is to find the action $\mathcal{A}_j$ of the matching rule $\mathcal{R}_j$, which has the highest priority in all the matching rules. It can be defined as follows:

$$\mathcal{A}_j \leftarrow Lookup(\hbar_i, \mathcal{T}_k)$$

And the lookup process of a packet header $\hbar_i$ against the multiple flow table $\mathcal{MT}$ is to find the action list:

$$< \mathcal{A}_{j1}, \mathcal{A}_{j2}, ..., \mathcal{A}_{jn} > \leftarrow Lookup(\hbar_i, \mathcal{MT})$$

The action $\mathcal{A}_{jk}$ will be *nil*, if the flow table $\mathcal{T}_k$ is skipped in the search path of the multiple flow table.

A cache $\mathcal{C}$ is a set of pairs $(\hbar_i, < \mathcal{A}_{j1}, \mathcal{A}_{j2}, ..., \mathcal{A}_{jn} >)$. If the information of packet header $\hbar_i$ has been stored in the cache $\mathcal{C}$, the lookup process will return its corresponding action list:

$$< \mathcal{A}_{j1}, \mathcal{A}_{j2}, ..., \mathcal{A}_{jn} > \leftarrow Lookup(\hbar_i, \mathcal{C})$$

Otherwise, the lookup process will return *nil*, which means a cache miss.

An update of a flow table can be done in one of three manners: insertion, deletion, and modification. Inserting a new rule $\mathcal{R}'_j$ and its associated action $\mathcal{A}'_j$ into a flow table can be represented as $\mathcal{T}' = \mathcal{T} + < \mathcal{R}'_j, \mathcal{A}'_j >$; Deleting an old rule $\mathcal{R}_j$ and its associated action $\mathcal{A}_j$ from a flow table can be represented as $\mathcal{T}' = \mathcal{T} - < \mathcal{R}_j, \mathcal{A}_j >$; Modifying an old rule $\mathcal{R}_j$ by replacing its associated action $\mathcal{A}_j$ with $\mathcal{A}'_j$ can be represented as $\mathcal{T}' = \mathcal{T} - < \mathcal{R}_j, \mathcal{A}_j > + < \mathcal{R}_j, \mathcal{A}'_j >$.

When an original rule $\mathcal{R}_j$ in the multiple flow table is deleted or modified, to guarantee the update consistency, we must evict all the pairs $(\hbar_i, < \mathcal{A}_{j1}, \mathcal{A}_{j2}, ..., \mathcal{A}_{jn} >)$ from the cache, where $\hbar_i \in \mathcal{R}_j$.

Our goal in this paper is to design and implement a cache mechanism, which has fast lookup speed, low memory consumption, high cache hit rate, as well as update consistence, to improve the lookup performance of the data plane and keep the data plane working while updating the flow tables.

III. FlowShadow: the Framework, Data Structures and Algorithms

A. The Framework of FlowShadow

FlowShadow achieves fast packet processing and supports uninterrupted update by caching microflows. The framework of an SDN datapath applying FlowShadow is illustrated in Figure 7, here we take the packet processing pipeline based on TCAM as an example of the original flow tables. There are two paths for packet processing: the fast path (FlowShadow) and the slow path (the original datapath based on the flow tables). When the system receives a packet, it first parses this packet and partitions the packet into the fields and the payload. The fields are handled by the datapath to find the rules that the packet should follow; and the payload is stored in the main memory waiting to be forwarded.

In the beginning, the packet's fields are searched in the FlowShadow to find the packet's corresponding microflow. If the microflow is found, the actions of the microflow will be executed on the packet; Otherwise, the fields will be searched in the original flow tables to find the rules and actions. Since not all of the rules are stored in the flow tables, it is possible that the packet cannot find any matching rule. If that happens, the switch will send a *packet-in* message to the controller. In the case of successfully finding the matching rules, FlowShadow will cache the fields (the microflow) and its associated data that includes the references of counters and the actions.

FlowShadow applies a hash table to store the microflows, as demonstrated in Figure 8. In the beginning, a 32-bit signature is generated from the fields; then the signature is hashed into a bucket that has a few entries for storing signatures and the pointers to the associated data. In the lookup process, the signature of the searched packet is compared with all the signatures stored in the same buckets to find the matching one. When the matched signature is found, to guarantee the correctness the packet's fields will be compared to the microflow's fields which are stored in the associated data. But in the insertion process, the signature and the pointer of the new microflow directly replaces the old signature and pointer according to the cache replacement policy.

To preserve the counters of rules and the ability of statistics, we store the references of counters in the microflow's associated data. Note that the counters of rules are organized as a pool for saving resources in both software switches [3] and hardware switches [13]. In this way, we not only keep the datapath having only one counter for one rule, but also can collect statistics of all packets regardless of processed by FlowShadow or the original flow tables.

Based on the observation that the amount of different actions is small, we store all the actions in a hash table (*Action Table*) as illustrated in Figure 8. When a new microflow is inserted into the hash table, FlowShadow first searches the Action Table to find the matching one. If there is a matching action in the Action Table, the reference of this action will be returned; otherwise, the new action is inserted into the Action

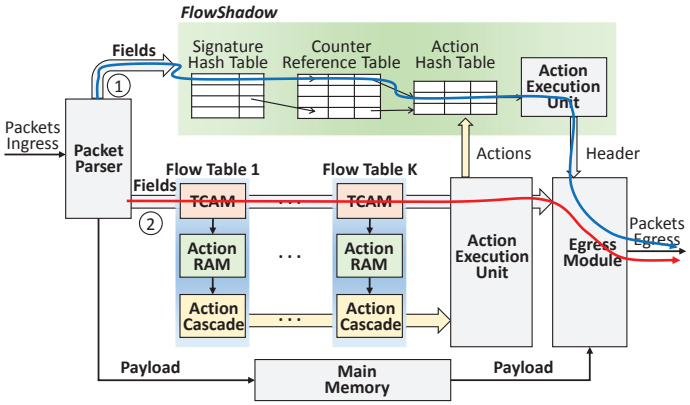


Fig. 7: The framework of an SDN datapath applying FlowShadow.

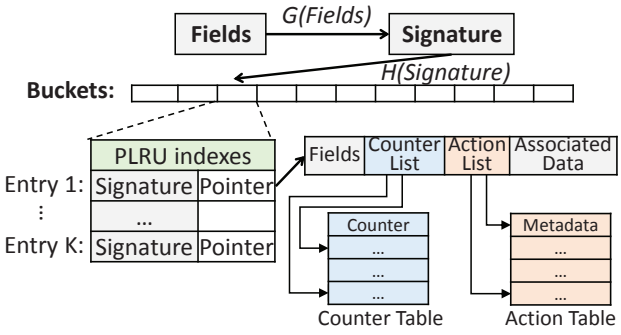


Fig. 8: The framework of FlowShadow.

Table, and the reference of this new action is returned. In the associated data of a microflow, the action list records the references of actions stored in the Action Table. By leveraging the Action Table, FlowShadow simultaneously reduces the memory consumption for storing actions and achieves update consistency which will be described in Section III-B3.

B. Update Consistency

The flow tables are dynamic. Rules are on-demand loaded into the flow tables according to the packets going through the SDN switches. When a rule is changed or removed from a flow table, its corresponding microflows cached in FlowShadow should be removed. Deleting all the microflows of a rule is difficult and time-consuming since the rule is the *superset* of the microflows. One way to remove the microflows of a rule is to scan the whole cache (hash table), but this solution is impractical to be implemented in high speed network devices. Another way is to record the relationship between the rule and its corresponding microflows in an extra hash table for fast updating. In FlowShadow, we apply the latter solution to address the mapping problem between rules and microflows. Specifically, we propose one method for strong update consistency (described in Section III-B1) and two methods for weak update consistency (described in Section III-B2 and Section III-B3).

1) An extra Signature Table for strong update consistency: Each rule contains a cookie (a 64-bit unsigned int) as an opaque identifier [5]. The cookie is specified by the controller when the rule is installed, and the cookie will be returned as part of each rule state and rule expired message.

We leverage these cookies as the keys of the *Signature Table* to record the mapping relationships between rules and microflows. Since a microflow matches various rules in the multiple flow table, FlowShadow also records the rules followed by this microflow into the Signature Table. When a rule is modified or removed, FlowShadow first looks up the rule's cookie against the Signature Table to get the microflows which follow this rule; then it removes all these microflows from the cache.

FlowShadow achieves strong update consistency by introducing the Signature Table. All the corresponding microflows stored in the cache are immediately removed when a rule is changed. However, removing the microflows of a modified rule is a burst of deleting operations for the cache. The performance of this solution is gradually decreasing as the cache size grows. For example, 0.05% of the microflows match to a rule on average when the flow table contains 2,000 rules. Assuming the cache can store 10 million microflows, 5,000 microflows should be removed from the cache when a rule is modified. Therefore, we further propose the weak update consistency solutions to smooth the burst introduced by the Signature Table.

2) An extra Cookie Table for weak update consistency: The number of rules in the flow tables is small, e.g., the commodity switches can store 1K∼20K rules [14]. As a rule and a cookie are a one-to-one correspondence, the number of different cookies is also small. Based on this observation, we can store all the cookies in a small hash table (*Cookie Table*) to indicate the states of rules. Meanwhile, we need to store the cookies in the microflow's associated data when it is inserted into the cache. FlowShadow looks up the Cookie Table to confirm whether this matching microflow is still valid. If it is invalid, the microflow will be removed from the cache and the packet will be searched in the flow tables.

The solution based on the Cookie Table has weak update consistency. The modification on a rule only impacts on the Cookie Table, and the microflows are not changed until packets belonging to these microflows are searched in the cache. Although the Cookie Table smoothes the update burst, it is the trade off between the lookup performance and the update performance. Each packet will be looked up against the Cookie Table a few times because a microflow may follow multiple rules. Despite the Cookie Table is small enough to be fitted in the onchip memory, FlowShadow still waste a lot of time on confirming the validities of the rules.

3) Reusing the Action Table for weak update consistency: In an SDN switch with a single flow table, the rule and the action is a one-to-one correspondence. The modification of a rule is directly reflected on its action. Meanwhile, the number of actions is very small. Based on these observations, we leverage the Action Table to implement update consistency.

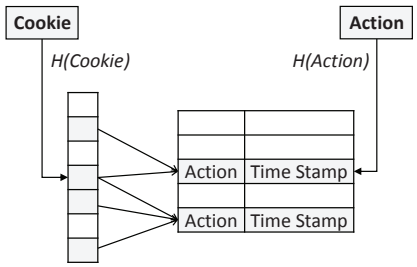


Fig. 9: The Cookie Table and the Action Table for weak update consistency.

When a microflow finds its corresponding action is invalid in the Action Table, it will be removed from the cache.

However, in the multiple flow tables, a final action stored in the Action Table is cascaded by the multiple middle actions of the rules. In other words, one action of a rule cannot be found in the Action Table anymore. To leverage the Action Table for update consistency, FlowShadow records each rule's corresponding final actions. We modify the Cookie Table to achieve this function by storing the references of actions instead of the signatures. Figure 9 demonstrates the details of the Action Table and the Cookie Table. When a rule is modified, FlowShadow can set the actions to invalid by looking up the Cookie Table to obtain the references of the actions. Note that, *compared to the above Cookie Table of signatures, the Action Table no longer requires FlowShadow to store the cookies in the associated data*.

Another problem should be addressed before implementing the Action Table in FlowShadow is to resolve the conflicts between the actions that have the same value. It is possible that an action inserted into the Action Table has the same value with an invalid action while its corresponding microflows in the cache are still unremoved. To resolve this conflict, we add the 64-bit time stamp in the entry of Action Table to identify the old action and the new action. Meanwhile, the associated data of a microflow in the cache also stores the time stamp that records the time of the microflow inserted in the cache. In this way, FlowShadow can identify the validity of a microflow according to its time stamp. If the microflow's time stamp is greater than the action's time stamp, it means this microflow is cached after the rules changed, therefore FlowShadow handles this packet according to the microflow's actions. Otherwise, FlowShadow deletes this microflow from the cache and send the packet to the flow tables for processing.

The Action Table is easy to maintain and is small enough to be fit into the on-chip memory. We can achieve weak update consistency without losing any performance by exploiting the Action Table in FlowShadow.

C. Cache Replacement Strategies

The number of microflows is much larger than the number of entries in the cache. For example, as illustrated in Figure 4(a), around 250K new flows are generated in one second in the campus traces. However, the cache only stores the temporally locality microflows. The microflows should be dynamically added into and removed from the cache to increase the cache hit rate and then improve the lookup speed.

Different with the data locality in the CPU, *the localities of microflows exist in the packets of the same microflow*. As the measuring results demonstrated in Figure 6(a) and Figure 6(b), more than 53.1% intra-flow packet intervals are less than 1,000 ms and more than 19.3% intra-flow packet intervals are less than 2 ms. Therefore, we only take the cache replacement strategies in each bucket to reduce the replacement cost and improve the performance.

On the other side, the lookup process should match all the entries of a bucket to find the exact matching one. Hence, to make the lookup process as fast as possible, FlowShadow only sets 1 entry, 2 entries, or at most 4 entries in one bucket, and it applies Pseudo-LRU [15] as the cache replacement strategy. After testing the lookup performance of FlowShadow with different numbers of entries in one bucket, we make a tradeoff between the cache hit rate and the lookup speed: *in the real implementation of FlowShadow, we set 1 entry in one bucket. In other words, we apply the direct replacement strategy in the cache*.

TABLE II: Hardware configuration.

Item	Specification
CPU	Intel Xeon E5645×2 (6 cores, 2 threads, 1.6GHz)
RAM	DDR3 ECC 48GB (1333MHz)
Motherboard	ASUS Z8PE-D12X (INTEL S5520)

TABLE III: The Internet backbone traces are captured from the routers of Chicago and San Jose in Mar. 20, 2014. [16]

Trace Name	BW (Gbps)	Time	Average Utilization	Total Flows
Backbone-1 (Chicago)	10	13:00∼13:05	14.21%	2,921,223
Backbone-2 (San Jose)	10	13:00∼13:05	32.58%	11,475,217

IV. Experiments

A. System Implementation

FlowShadow is implemented based on the open source project Open vSwitch (release version 2.1.2 [6]). Specifically, we have done the following works: 1) We modify the *fast path* of OVS' datapath to implement the cache mechanism and the update scheme of FlowShadow; 2) We implement a module for supporting multiple flow tables by reusing the existing modules of the single flow table in OVS. This multiple hash table module still has 16-bit port numbers (Starting from OpenFlow 1.1, the later OpenFlow version uses 32-bit port numbers), and it can support the most of actions defined in OpenFlow 1.4; 3) For measuring the performance, the FlowShadow runs in the user space instead of the kernel space.

B. Experiments Setup

1) Computational platform: The OVS with FlowShadow runs on a commodity PC with two CPUs (6-core, 1.6 GHz per core). Relevant hardware configuration is listed in Table II.

The PC runs Linux Operating System in the version 2.6.41.9-1.fc15.x86_64. The part of multi-core parallel processing is developed using OpenMP API [17] in version 2.5.

2) Flow tables: The multiple flow table in our experiments consists of two real flow tables: 1) an IP prefix table downloaded from RIPE [18] (Equinix, 2012-01-01) contains 388,344 IP prefixes; 2) and an ACL set from the website [19] contains 752 rules that consider 5-tuple flows and subject to the format of ClassBench [20].

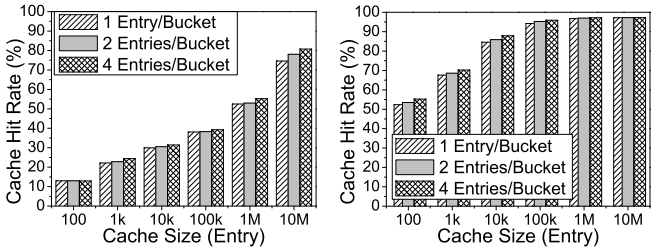
3) Traces: The traces used in our experiments are captured from 3 different networks: campus network, enterprise network, and Internet backbone. Each trace sustains 5 minutes, and the details are listed in the Table I and Table III. The campus traces and the backbone traces have packet information, and the enterprise traces only have the flow information. Therefore, in the experiments we test the performance of FlowShadow under the campus traces and the backbone traces. Because of the limited space and the similar results of traces with the same type of network, *we choose one trace from each type of network to demonstrate the experimental results.*

C. Experimental results

1) The cache hit rate: To measure the cache hit rate, in the experiments we set the total number of entries in the cache as the product of the number of buckets and the number of entries in one bucket, i.e, different methods have the same cache size. PLRU [15] is applied as the cache replacement strategy in FlowShadow when the bucket size is 2 or 4; and the direct replacement strategy is used in FlowShadow when the bucket size is 1. The experimental results of the cache hit rates are demonstrated in Figure 10(a) and Figure 10(b), respectively. The performance of the direct replacement strategy is close to the performance of PLRU. For example, under the Backbone-1 trace, FlowShadow with 100K entries in the cache achieves 94.22% and 95.9% cache hit rates with the direct replacement strategy and the PLRU, respectively.

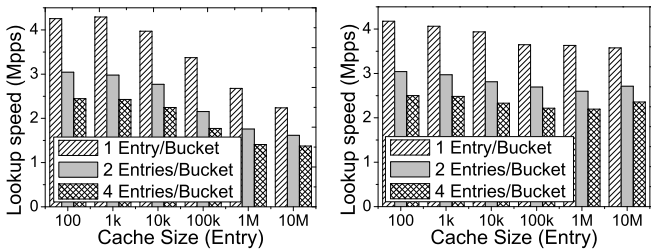
The Backbone-1 trace presents better network locality than the Campus-1 trace. When the cache has 10M entries, the Backbone-1 trace achieves around 97.22% cache hit rate, but the Campus-1 trace only achieves 74.59% cache hit rate. Digging more deeply, as illustrated in Figure 4(a), we discover that about 66.2% microflows have 1 or 2 packets and about 93.5% microflows have less than 10 packets in the Campus-1 traces. Consequently, FlowShadow has better performance in the scenarios that have more elephant flows, such as the backbone network [10], the enterprise network [11], and the data center network [21].

2) The lookup speed: The experimental results of the lookup speeds of FlowShadow with different settings are demonstrated in Figure 11(a) and Figure 11(b), respectively. Note that the lookup speeds shown in Figure 11(a) and Figure 11(b) are the speeds of FlowShadow (i.e., the fast path), and the speeds of the system including the fast path and the slow path will be illustrated in Figure 16(a) and Figure 16(b), respectively. FlowShadow with 100 entries in the cache (1 entry per bucket) achieves 4.26 Mpps (million packets per second) and 4.18 Mpps under the Campus-1 and Backbone-1 traces, respectively. The lookup performance of FlowShadow decreases as the cache size grows because FlowShadow runs on the commodity CPU and the memory access performance of CPU drops down as the cache size grows.



(a) The Campus-1 trace. (b) The Backbone-1 trace.

Fig. 10: The cache hit rates of FlowShadow with different cache replacement strategies and cache sizes under the Campus-1 and Backbone-1 traces.



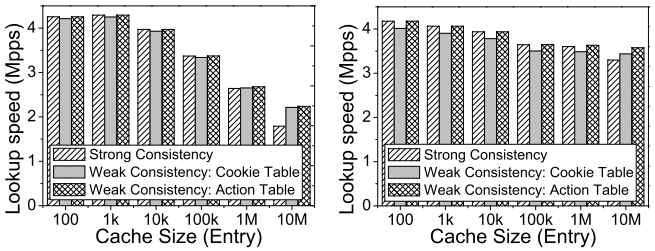
(a) The Campus-1 trace. (b) The Backbone-1 trace.

Fig. 11: The lookup speeds of FlowShadow with different cache replacement strategies and cache sizes under the traffics of Campus-1 and Backbone-1.

FlowShadow searches all entries in a bucket to find the matching signature, hence the lookup speed of FlowShadow decreases as the bucket size grows. As illustrated in Figure 11(a) and Figure 11(b), under the Campus-1 trace, FlowShadow with 1K entries in the cache achieves 4.29 Mpps, 2.98 Mpps and 2.45 Mpps when the bucket sizes are 1, 2 and 4, respectively. Since the larger bucket sizes improve the cache hit rates by a small amount, we apply the directly replacement strategy in FlowShadow to reduce the lookup complexity and improve the lookup performance.

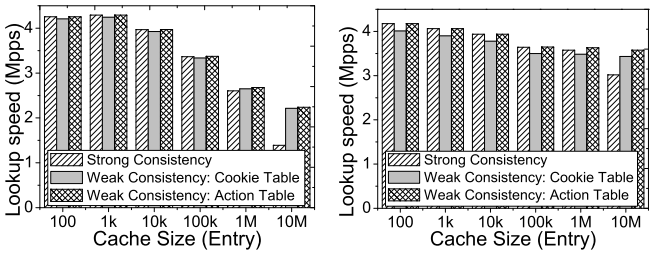
3) Scalability: Another factor that affects the lookup performance is the scheme of update consistency. Figure 12(a) and Figure 12(b) show the lookup speeds of FlowShadow with different settings and different schemes for update consistency. From Figure 12(a) and Figure 12(b), we can conclude that the Action Table achieves the highest performance in the three schemes proposed by us in Section III-B.

The lookup performances of FlowShadow with different update consistency schemes are also impacted by the number of flow tables. To present the good scalability of FlowShadow applying the Action Table for update consistency, we conducts the experiments with different flow tables under the Campus-



(a) The Campus-1 trace. (b) The Backbone-1 trace.

Fig. 12: The lookup speeds of FlowShadow with different update consistency schemes and cache sizes under the traffics of Campus-1 and Backbone-1. (Single flow table and 10% rules are modified per second)



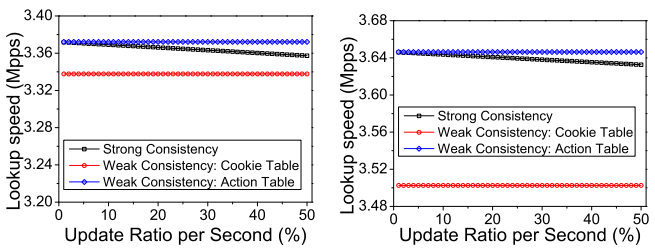
(a) The Campus-1 trace. (b) The Backbone-1 trace.

Fig. 13: The lookup speeds of FlowShadow with different update consistency schemes and cache sizes under the traffics of Campus-1. (Two flow tables and 10% rules of each flow table are modified per second)

1 trace and the Backbone-1 trace. The experimental results are demonstrated in Figure 12(a), Figure 12(b), Figure 13(a) and Figure 13(b). The Action Table and the strong update consistency keep the lookup speeds with different flow tables. However, the lookup performance of the Cookie Table gradually drops as the number of flow tables grows, and the lookup performance of strong consistency scheme dramatically drops down as the cache size grows and the number of flow tables increases.

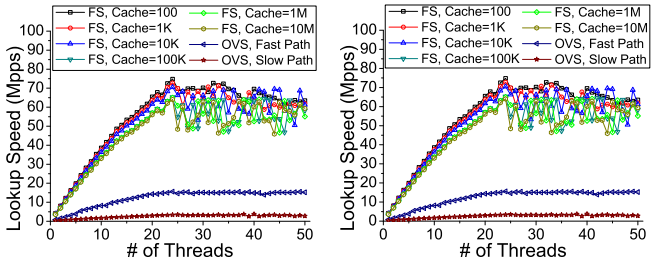
The update speeds of the flow tables also effect the lookup performance of FlowShadow with different update consistency schemes. From the experimental results shown in Figure 14(a) and Figure 14(b), we can conclude that the update schemes (the Cookie Table and the Action Table) of weak consistency have stable performance with different update speeds; and the update scheme of strong consistency has poorer lookup performance when there is higher update speed.

4) The system performance: The system applies FlowShadow as the fast path runs on the commodity PC that contains 2 CPUs (total 24 physical threads). Therefore, we can speed up the lookup of the system by increasing the number of threads. Figure 15(a) and Figure 15(b) demonstrate the experimental results. From Figure 15(a) and Figure 15(b), we can see that the system achieves the highest performance with 24 work threads because more threads will lead to the



(a) The Campus-1 trace. (b) The Backbone-1 trace.

Fig. 14: The lookup speeds of FlowShadow with different update speed under the traffics of Campus-1 and Backbone-1. (1 flow table and cache size=100K)



(a) The Campus-1 trace. (b) The Backbone-1 trace.

Fig. 15: The lookup speed of FlowShadow with different update speeds under the traffics of Campus-1 and Backbone-1. (1 flow table and cache size=100K)

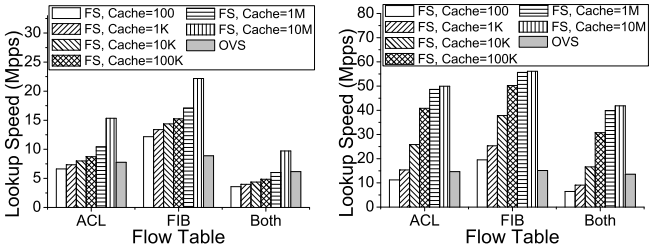
competition among threads and waste more time on the context switching.

The lookup performances of the system with FlowShadow and without FlowShadow on different flow tables are illustrated in Figure 16(a) and Figure 16(b), respectively. Under the Backbone-1 trace, the system with FlowShadow (10M entries in the cache) achieves 49.94 Mpps, 56.11 Mpps and 41.86 Mpps on the flow tables of ACL, FIB, and both ACL and FIB, respectively. It is about 3.4× of the original OVS which achieves 14.67 Mpps, 15.11 Mpps and 13.58 Mpps on the flow tables of ACL, FIB, and both.

V. Related Work

A. Incremental Update for SDN

Except Maple [22], most SDN program languages and compilers (e.g., Frenetic [23], NetCore [24], and NetKAT [25]) do not provide any support for incremental policy updates. They simply compile the new policies and replace the rules in the flow tables of switches. Consequently, a considerable number of unnecessary rule updates still have to be conducted. Maple improves the incremental update performance by introducing tree-style abstraction to support incremental flow table updates. The compiler of Maple, however, still makes a large amount of priority updates due to the consecutive priority values. Xitao *et al.* [26] further reduce the redundant rule updates by extending policy compiler to build rule dependency along with the compilation.



(a) The Campus-1 trace. (b) The Backbone-1 trace.

Fig. 16: The lookup performance of the methods with different flow tables under the traffics of Campus-1 and Backbone-1 (24 threads).

Besides the works on supporting incremental updates in compilers, many recent works provide solutions to maintain different consistency properties during network updates. Reitblatt *et al.* [27] develop a theoretical model to capture the essential behaviors of SDNs and propose two level (per-packet and per-flow) consistency as canonical to maintain packet coherence. McGeer *et al.* [28] conserve rule space while keeping update consistency by exploiting the identity of switch rules with Boolean functions to ensure that only a single set of rules is present on a switch at any time. Katta *et al.* [29] propose new algorithms that trade the time required to perform a consistent update against the rule space overhead required to implement it. Dionysus [30], developed from the general architecture [31] for consistent updates that separates the twin concerns of consistency and efficiency, is a system fast consistent network updates. Dionysus dramatically improves the update speed by dynamic scheduling the updates.

Architecturally, FlowShadow, providing the support for continuous operations while updating the flow tables, is orthogonal but complementary to the aforementioned update approaches for SDN that can reduce the redundant rule updates, reduce rule space, improve update speed, as well as keep update consistency.

B. Exploiting Cache in Packet Processing

Exploiting the temporal locality within the flow of network packets to optimize resource utilization is not new [32]. Based on the observation that network communications exhibit strong locality, prior works on IP Lookup [33], [34], [35], [36] cache a small number of IP prefixes in the onchip memory to improve the lookup performance. However, the lookup processes of these schemes still need to subject to longest prefix match in the cache. To speed up the matching process in the cache, some works [37], [38], [39], [40], [41] cache the destination addresses in fast memories. All of them deal with only one forwarding table, none of them can tackle complex multiple flow tables.

Cache in SDN devices. DIFANE [42] advocates caching of ternary rule and supports multiple flow table. It relegates the controller to the simple task of partitioning rules over the physical switches, and it selectively directs packets through intermediate switches that store the necessary rules. The side effect of DIFANE is that the controller loses the global visibility of flow states. A similar idea with DIFANE, proposed by Lee *et al.* [43], deploys flow cache servers between controller and switches. A flow cache server stores the microflows to reduce the number of packet-in messages to controller, but it breaks the update consistency. In opposite directions, CacheFlow [44] takes the hardware switch as a cache and leverages the software switch to store the large flow tables. However, the cost of cache miss is very large since replacing rules in TCAMs is slow and complex.

Cache in a single SDN device. Congdon *et al.* [2] use a prediction circuit before flow table to speedup flow classification of incoming packets. In essence, the prediction circuit, only stores the forwarding information, is a cache for microflows. However, the prediction circuit cannot preserve rule counters and does not address the issue of statistics. Open vSwitch [3] (OVS) is an open-source software implementation of an OpenFlow switch, principally designed to work as a virtual switch in virtualized environments. OVS implements two datapaths, the *fast path* and the *slow path*, to process the incoming packets. The fast path is the cache of exact-match rules in the kernel space, and the slow path is the original flow table in the user space. The first packet of each microflow undergoes slow path to identify the highest-priority matching wildcard rule [4], and the subsequent packets of the microflows in the cache are processed by the fast path. OVS's fast path stores the reference of rules in the microflows' associated data to preserve the rule counters and keep update consistency, but it only supports the single flow table (e.g., OpenFlow 1.0 [5]) and its lookup performance is poor since each packet still needs to match the rule to get the final action.

Summarily, FlowShadow addresses the issues of keeping update consistency, preserving rule counters, achieving fast lookup, and supporting multiple flow tables. None of the aforementioned works can handle all of these issues well.

VI. Discussion and Conclusion

Discussion: In the current version of FlowShadow, when a rule is deleted from the flow table, its corresponding microflows are all deleted from the cache too. But in the most time, a rule is evicted from the flow table because of the limited memory space of a switch, and the rule itself has not been modified. In this case, to improve the lookup performance and reduce the rule switching frequency, FlowShadow can keep the rule's microflows in the cache until the entries are replaced by other microflows. To correctly identify the sematic between the rule deletion and the rule eviction is our future work.

FlowShadow has simple logic and costs a little resource. It is easy to be implemented as a small module on the switch chip, such as Intel FM6000 [13], which has 24K entries of 36-bit TCAM and 64K binary search tree. As illustrated in Figure 10(a) and Figure 10(b), the cache with 1K entries can achieve 67.59% hit rate under the Backbone-1 trace, i.e., 67.59% packets can be processed by the FlowShadow, which

not only improves the lookup performance, but also reduces the power consumption of the TCAM.

Conculsion: In this paper, we propose FlowShadow, a cache based architecture for improving the lookup performance of the datapath and supporting the datapath ongoing working while updating the flow tables. By applying the Action Table, FlowShadow achieves weak update consistency and obtains good update performance. Massive experiments are conducted and the experimental results demonstrate that FlowShadow has high lookup and update performance, and good scalability at different update speeds and with different numbers of flow tables.

REFERENCES

[1] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[2] P.T. Congdon, P. Mohapatra, M. Farrens, and V. Akella. Simultaneously reducing latency and power consumption in openflow switches. *IEEE/ACM Transactions on Networking*, 22(3):1007–1020, June 2014.

[3] Ben Pfaff, Justin Pettit, Keith Amidon, Martin Casado, Teemu Koponen, and Scott Shenker. Extending Networking into the Virtualization Layer. In *HotNETs'09*.

[4] Gaetano Catalli. Open vSwitch: performance improvement and porting to FreeBSD.

[5] OpenFlow Switch Specification 1.0. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/op-enflow-spec-v1.0.0.pdf, 2014. [Online].

[6] Open vSwitch 2.1.2. http://openvswitch.org/releases/openvswitch-2.1.2.tar.gz, 2014. [Online].

[7] Slim Ben Fredj, Thomas Bonald, Alexandre Proutire, G. Rgni, and James W. Roberts. Statistical bandwidth sharing: a study of congestion at flow level. In *SIGCOMM'01*, pages 111–122, 2001.

[8] Yin Zhang, Lee Breslau, Vern Paxson, and Scott Shenker. On the characteristics and origins of internet flow rates. *Computer Communication Review*, 32:309–322, 2002.

[9] Tatsuya Mori, Masato Uchida, Ryoichi Kawahara, Jianping Pan, and Shigeki Goto. Identifying elephant flows through periodically sampled packets. In *Internet Measurement Workshop*, pages 115–120, 2004.

[10] Lin Quan and John Heidemann. On the characteristics and reasons of long-lived internet flows. In *IMC'10*, pages 444–450, 2010.

[11] Aiyou Chen, Yu Jin, Jin Cao, and L. E. Li. Tracking Long Duration Flows in Network Traffic. In *IEEE INFOCOM*, pages 206–210, 2010.

[12] Peyman Kazemian, George Varghese, and Nick McKeown. Header Space Analysis: Static Checking for Networks. In *NSDI'12*, pages 113–126, 2012.

[13] Intel Ethernet Switch FM6000 Series - Software Defined Networking. http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ethernet-switch-fm6000-sdn-paper.pdf, 2014. [Online].

[14] Brent Stephens, Alan Cox, Wes Felter, Colin Dixon, and John Carter. PAST: Scalable Ethernet for data centers. In *CoNext'12*, pages 49–60. ACM, 2012.

[15] Pseudo-LRU. http://en.wikipedia.org/wiki/Pseudo-LRU, 2014. [Online].

[16] CAIDA Data. http://www.caida.org/data/, 2014. [Online].

[17] The OpenMP API specification for parallel programming. http://openmp.org/wp/. http://openmp.org/wp/, 2014. [Online].

[18] RIPE NCC: RIPE Network Coordination Centre. http://www.ripe.net/, 2014. [Online].

[19] An Access Control List. http://www.arl.wustl.edu/~hs1/project/filterset/acl1, 2014. [Online].

[20] David E Taylor and Jonathan S Turner. Classbench: A packet classification benchmark. In *INFOCOM'05*, pages 2068–2079. IEEE, 2005.

[21] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The nature of data center traffic: measurements & analysis. In *IMC'09*, pages 202–208. ACM, 2009.

[22] Andreas Voellmy, Junchang Wang, Y Richard Yang, Bryan Ford, and Paul Hudak. Maple: Simplifying sdn programming using algorithmic policies. In *SIGCOMM'13*, pages 87–98. ACM, 2013.

[23] Nate Foster, Rob Harrison, Michael J Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker. Frenetic: A network programming language. 46(9):279–291, 2011.

[24] Christopher Monsanto, Nate Foster, Rob Harrison, and David Walker. A compiler and run-time system for network programming languages. *ACM SIGPLAN Notices*, 47(1):217–230, 2012.

[25] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. Netkat: Semantic foundations for networks. In *SIGPLAN'14*, pages 113–126. ACM, 2014.

[26] Xitao Wen, Chunxiao Diao, Xun Zhao, Yan Chen, Li Erran Li, Bo Yang, and Kai Bu. Compiling Minimum Incremental Update for Modular SDN Languages. In *HotSDN'14*. ACM, 2014.

[27] Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, and David Walker. Abstractions for network update. In *SIGCOMM'12*, pages 323–334. ACM, 2012.

[28] Rick McGeer. A safe, efficient update protocol for OpenFlow networks. In *HotSDN'12*, pages 61–66. ACM, 2012.

[29] Naga Praveen Katta, Jennifer Rexford, and David Walker. Incremental Consistent Updates. In *HotSDN'13*, pages 49–54. ACM, 2013.

[30] Xin Jin, Hongqiang Harry Liu, Rohan Gandhi, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Jennifer Rexford, and Roger Wattenhofer. Dynamic Scheduling of Network Updates. In *ACM SIGCOMM*, 2014.

[31] Ratul Mahajan and Roger Wattenhofer. On consistent updates in software defined networks. In *HotNET'13*. ACM, 2013.

[32] Raj Jain and Shawn Routhier. Packet trains–measurements and a new model for computer network traffic. *IEEE Journal on Selected Areas in Communications*, 4(6):986–995, 1986.

[33] David C Feldmeier. Improving gateway performance with a routing-table cache. In *INFOCOM'88*, pages 298–307. IEEE, 1988.

[34] Changhoon Kim, Matthew Caesar, Alexandre Gerber, and Jennifer Rexford. Revisiting route caching: The world should be flat. In *Passive and Active Network Measurement*, pages 3–12. Springer, 2009.

[35] Zhuo Huang, Gang Liu, and Jih-Kwon Peir. Greedy prefix cache for IP routing lookups. In *ISPAN'09*, pages 92–97. IEEE, 2009.

[36] Nadi Sarrar, Steve Uhlig, Anja Feldmann, Rob Sherwood, and Xin Huang. Leveraging Zipf's law for traffic offloading. *ACM SIGCOMM Computer Communication Review*, 42(1):16–22, 2012.

[37] Chi-Cheng Wu, Jia-Long Wu, and Huang-Sen Chiu. Improving IP lookup Performance with a Routing-Table Cache. *IEEE Communications Letters*, 7(3), 2003.

[38] Soraya Kasnavi, Paul Berube, Vincent Gaudet, and José Nelson Amaral. A cache-based internet protocol address lookup architecture. *Computer Networks*, 52(2):303–326, 2008.

[39] Yi Wang, Boyang Xu, Dongzhe Tai, Jianyuan Lu, Ting Zhang, Huichen Dai, Beichuan Zhang, and Bin Liu. Fast name lookup for named data networking. In *Quality of Service (IWQoS), 2014 IEEE 22nd International Symposium of*, pages 198–207. IEEE, 2014.

[40] Yi Wang, Tian Pan, Zhian Mi, Huichen Dai, Xiaoyu Guo, Ting Zhang, Bin Liu, and Qunfeng Dong. Namefilter: Achieving fast name lookup with low memory cost via applying two-stage bloom filters. In *INFOCOM, 2013 Proceedings IEEE*, pages 95–99. IEEE, 2013.

[41] Yi Wang, Keqiang He, Huichen Dai, Wei Meng, Junchen Jiang, Bin Liu, and Yan Chen. Scalable name lookup in ndn using effective name component encoding. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 688–697. IEEE, 2012.

[42] Minlan Yu, Jennifer Rexford, Michael J Freedman, and Jia Wang. Scalable flow-based networking with DIFANE. *ACM SIGCOMM Computer Communication Review*, 40(4):351–362, 2010.

[43] Bu-Sung Lee, Renuga Kanagavelu, and Khin Mi Mi Aung. An efficient flow cache algorithm with improved fairness in Software-Defined Data Center Networks. In *CloudNet'13*, pages 18–24. IEEE, 2013.

[44] Naga Katta, Jennifer Rexford, and David Walker. Infinite CacheFlow in Software-Defined Networks. In *HotSDN'14*. ACM, 2014.