IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.Doi Number

Efficient Differentiated Storage Architecture for Large-scale Flow Tables in Software-Defined Wide-Area Networks

Bing Xiong[1,2], Rengeng Wu[1], Jinyuan Zhao[3], Zhuofan Liao[1], Jin Wang[1]

[1]School of Computer & Communication Engineering, Changsha University of Science & Technology, Changsha, 410114 P. R. China
[2]Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122 USA
[3]School of Computer Science and Engineering, Central South University, Changsha 410075 P. R. China

Corresponding author: Bing Xiong (e-mail: xiongbing@csust.edu.cn).

This work was supported in part by National Natural Science Foundation of China (61502056), Hunan Provincial Natural Science Foundation of China (2015JJ3010), Scientific Research Fund of Hunan Provincial Education Department (18B162, 15B009), and Scientific Research Innovation Fund of Hunan Postgraduates (CX2018B567).

ABSTRACT As a novel network paradigm, Software Defined Networking (SDN) decouples control logic functions from data forwarding devices, and introduces a separate control plane to manipulate underlying switches via southbound interfaces like OpenFlow. This paradigm offers numerous benefits for wide area networks (WAN), like promoting application performance and reducing deployment costs, but poses serious challenges on the storage resources and lookup performance of large-scale flow tables in OpenFlow switches. This paper is thus motivated to propose an efficient differentiated storage architecture for large-scale flow tables in OpenFlow-based software-defined WAN. Firstly, we investigate into the impact of wildcards in match fields on the packet-in-batch feature within a flow based on network traffic locality. Then, packet flows are dynamically distinguished into active ones and idle ones in terms of their short-term states. Subsequently, we store the match fields of active flows and idle flows respectively in TCAM and SRAM, and the content fields of both types of flows in DRAM, to effectively relieve the insufficiency of TCAM capacity. Finally, we evaluate the performance of our proposed flow table storage architecture with real network traffic traces by experiments. The experimental results indicate that our proposed storage architecture with the active/idle flow differentiation obviously outperforms the traditional one applying the elephant/mice flow differentiation in terms of TCAM hit rates and average flow table access time.

INDEX TERMS Software-Defined WAN, OpenFlow Switches, Large-scale Flow Tables, Differentiated Storage Architecture, Active/Idle Flow Differentiation

I. INTRODUCTION

As an emerging network paradigm, Software-Defined Networking (SDN) separates network control functions from data forwarding devices, and allows for a separate controller entity to manage and control all underlying switches through southbound interface typically OpenFlow [1]. This novel paradigm paves the way for a more flexible, programmable, and innovative networking, and is commonly considered as one of the promising directions towards future wide area networks (WAN) [2], including data center interconnections [3], energy Internet, smart grid and industry Internet. The OpenFlow-based SDN concept provides numerous benefits to WANs, like promoting data transfer efficiency, improving application performance, and reducing deployment costs. As a pioneer enterprise, Google designed and implemented a private software-defined WAN (SD-WAN) called B4, connecting its data centers across the planet as early as 2013 [4]. After that, B4 incrementally moved from offering best-effort content-copy services to carrier-grade availability for the following 5-year evolution [5].

With the evolution of OpenFlow protocol versions, the number of match fields in a flow entry has increased from 12 in version 1.0 [6] to 44 in version 1.5 [7], which leads to a significant increase in the size of a flow entry. Meanwhile, there will be a large number of flow entries corresponding to simultaneous flows for large-scale SDN deployments, especially in wide area networks. The above two factors will produce a multiplication effect on the space occupation of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2942208, IEEE Access

the flow tables, and lead to a sharp increase in the requirements of flow table storage resources in OpenFlow switches. To date, TCAM is the most prevalent medium to store flow tables, as it supports wildcarding and outputs lookup results in one clock cycle. However, its capacity is limited due to its expensive cost, high power consumption and low integration, and is difficult to meet the storage space requirements of large-scale flow tables. Consequently, it is indispensable to build a delicate storage architecture for large-scale flow tables in OpenFlow switches.

Until now, the most popular solution to the above problem is to accommodate flow tables by combining TCAM with other media such as SRAM. An initial representative work from the research group led by Nick McKeown [19] [20], designed and implemented an OpenFlow switch on the NetFPGA platform, which distinguished flow entries in terms of their match fields into ones with wildcards and ones with all exact data, and stored them respectively in TCAM and SRAM. This design guarantees fast flow table lookups, as TCAM supports direct content matching with wildcards and SRAM also can achieve efficient lookups on exact flow tables typically by hashing. However, the match fields of a flow entry generally tend to carry wildcards, primarily as there are mutex relationships even between some required match fields such as IPV4_SRC/IPV4_DST vs IPV6_SRC/IPV6_DST, TCP_SRC/TCP_DST vs UDP_SRC/UDP_DST. Therefore, the number of flow entries with wildcards is prone to exceed TCAM capacity especially for large-scale SDN deployments. To tackle this problem, several researchers divided flows into elephant ones and mice ones, respectively accommodated in TCAM and SRAM [23][24]. They took advantage of long-term distribution characteristics of packet traffic over flows, but there is still room for improvement due to their negligence of flow dynamics.

For the above problems, this paper is motivated to devise an efficient storage architecture for large-scale flow tables in OpenFlow-based wide area networks. In particular, the architecture is built by exploiting the accessing properties of TCAM/SRAM/DRAM, and the short-term characteristics of packet flows. This paper is an extension of our previous work presented in the 21st IEEE International Conferences on High Performance Computing and Communications (HPCC2019). Different from our previous work, this paper focuses on the scenario of software-defined wide-area networks, estimate TCAM hit rates of different major/minor flow differentiation method based on the measurements of their parameters with real traffic traces, and quantizes the impact of wildcards in the match fields on the packet-in-batch property of a flow by traffic measurements. Moreover, we also reset typical parameter values to provide a more clear comparison between flow table storage architectures with different major/minor flow differentiation method. The main contributions of this paper are summarized as follows.

- Building a differentiated storage architecture for large-scale flow tables, by accommodating the match fields of major flows and minor ones respectively in TCAM and SRAM, and the content fields of both types of flows in DRAM, which effectively solves the storage space problem of large-scale flow tables.
- Having an investigation into the property of packet arrivals in batches within a traditional flow based on network traffic locality, and taking an insight into the impact of wildcards in the match fields on the packet-in-batch property of a flow specified by OpenFlow.
- Proposing a novel major/minor flow differentiation method that distinguishes packet flows into active ones and idle ones in terms of short-term flow states based on packet-in-batch arrivals within a flow, which gains higher TCAM hit rates in the above flow table storage architecture.
- Formulating the TCAM hit rates for flow table storage architecture applying our proposed active/idle flow differentiation method and the existing elephant/mice flow one, and comparing between both differentiation methods in terms of TCAM hit rates and average flow table access time by experiments with real network traffic traces.

The remainder of this paper is organized as follows. Section II introduces related work. In Section III, we design a differentiated storage architecture of large-scale flow tables based on the accessing features of their storage media. Section IV analyzes the traditional elephant/mice flow differentiation method, investigates into the packet-in-batch property of a flow, and proposes the active/idle flow differentiation method. In Section V, we describe the algorithmic implementations of our proposed flow table storage architecture. Section VI evaluates the performance of our proposed flow table storage architecture in terms of TCAM hit rates and average flow table access time by experiments with real network traffic traces. Finally, we conclude the paper in Section VII.

II. RELATED WORKS

Due to the limitation of flow table space at OpenFlow switches, packet forwarding may suffer from scarce storage resources and degraded lookup performance. Extensive studies have been carried out to reduce the flow table size, primarily through the following approaches: rule placement optimization, flow table aggregation, flow table compression, and flow table storage/lookup architecture design.

Rule placement is optimized to minimize the number of installed flow rules, from the global view of OpenFlow network by automatically rerouting important flows like the elephant one [8][9], or single OpenFlow switch by utilizing the characteristics of hardware and software flow tables to make decision of new flow installation [10]. Rule placement optimization increases flow table utilization, but still cannot cope with a large number of simultaneous flows especially in large-scale SDN deployments. Flow table aggregation

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2942208, IEEE Access

aims to reduce the amount of flow entries by substituting a set of overlapping flow rules with a more generalized one [11][12], or by merging a set of minor flow entries into a few ones through rule restruction [13]. However, this method will be out of action for flow entries without identical prefixes, and need to consider security concerns while placing wildcard rules at the switches [14]. Flow table compression shrinks redundant match fields into shorter identifier or labels with forwarding information, like Flow-ID [15], VLAN-ID [16] and Tag-in-Tag [17], or splits each bloat flow table into multiple sub-flow tables in terms of the coexistence and conflict relationships among fields [18]. Flow table compression has effectively reduced flow entry size, but have increased the complexity and overheads of flow table lookups.

To date, much work has been done on large-scale flow table storage architecture. Naous et al. [19][20] described the implementation of an OpenFlow Switch on the NetFPGA platform, where the flow table used a combination of on-chip TCAM and off-chip SRAM to support a large number of flow entries. In particular, TCAM and SRAM are respectively responsible for accommodating wildcard flows and exact flows for fast flow table lookups. Similarly, Matsumoto et al. [21][22] designed GPU-accelerated flow switching architecture, which allocates an exact flow table and a wildcard-enabled flow table respectively in host and GPU memories, to tackle massive flow entries. Unfortunately, the required storage space of flow entries with wildcards is liable to go beyond the TCAM capacity, especially for supporting high versions of OpenFlow protocols in large-scale SDN deployments. Lee et al. [23] proposed a differentiated flow cache framework with dynamic-index hashing for placements and a localized LRU-based replacement strategy, based on the observation from data center traffic that elephant flows are very large in size (data volume) but few in numbers when compared to mice flows. Similarly, Katta et al. [24] implemented a hybrid switch by utilizing the benefits of hardware and software for flow table storage, where heavy-hitter flows were installed at TCAM for fast processing and the remaining flows were placed at the main memory, in terms of the characteristics that traffic tends to follow a Zipf distribution. They increased the cache hit ratio and achieved fast flow table lookups, but there still is room for improvement due to their lack of consideration on flow dynamics.

Furthermore, Ding et al. [25] proposed a hybrid flow table storage architecture, utilizing NVM-based TCAM to cache the most popular rules to improve the cache-hit ratio, while relying on a very small-size SRAM-based TCAM to handle cache-miss traffic to decrease update latency. Wang et al. [26] built a traffic-aware hybrid rule allocation scheme by logically split TCAM into two parts: reactive and proactive, which can be dynamically adjusted according to network traffic behavior. Cheng et al. [27] synthesizes TCAM compatible entries by using binary decision trees and employs SRAM for further comparisons, to significantly reduce TCAM consumption and fulfill low power consumption. Lee et al. [28] presented a novel low-latency bundle-updatable TCAM scheme, which transforms the original ternary rule data into a binary code word by binary tree-based prefix encoding, and determines the range of overlap in SRAM addresses to facilitate updating. This greatly decreases latency in cases where multiple rules are required to update on an SRAM-based TCAM. However, these storage architectures cannot meet the storage space requirements of large-scale flow tables on account of the limited capacity of TCAM and SRAM.

Some researchers have concentrated on flow table lookup architecture in OpenFlow networks. Matsumoto et al. [21][29] proposed a flow table lookup framework LightFlow, by introducing two-dimensional parallelization of the linear search on the wildcard-aware flow table, and automatically updating the hash-based exact flow table based on the result of the wildcard-aware table lookup, to speed up the packet switching process. Ferkouss et al. [30] implemented a pipeline lookup architecture based on a 100Gig network processor platform, which extended the recursive flow classification by combining TCAM with the wildcard match and SRAM with the exact match to enhance packet classification performance. Li et al. [31][32] proposed a hybrid lookup scheme integrating multiple-cell hash table with TCAM for flow table matching, to simultaneously reduce the cost and power consumption of lookup structure without sacrificing the lookup performance. Nevertheless, these lookup architectures did not achieve a satisfactory performance without adequate consideration of network traffic characteristics.

III. LARGE-SCALE FLOW TABLE STORAGE

A. Software-Defined Wide Area Networks

The increasing number of Internet users, smart phone and other data collection terminals have generated an increasing amount of network traffic. The massive traffic needs to be transferred to other network endpoints across wide area networks. This puts forward higher demands on network performance including throughput, scalability, manageability, and flexibility. By introducing the concept of software-defined networking, a WAN will obtain many technical advantages, including intelligent path control, application performance optimization, automatic configuration and management. Consequently, the emerging paradigm of software-defined WAN will become a significant trend for enterprises to acquire better transmission service, save deployment cost, and release human resources [4][5].

Fig.1 demonstrates a typical SD-WAN deployment scenario. As shown in the scenario, OpenFlow switches connect a variety of network entities, such as cloud data centers, enterprise networks and even energy networks, to the Internet. These connections are implemented through various

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2942208, IEEE Access

kinds of high-speed links including MPLS, Ethernet, xDSL, SDH, 4G/5G LTE. Logically centralized control plane takes charge of all OpenFlow switches, and provides the function of topology management, path computation, data security and policy control over global networks. Eventually, network operators can achieve a bunch of network management functions, typically user access control, traffic visualization, service orchestration, and big data analysis, through programming interface provided by the control plane.

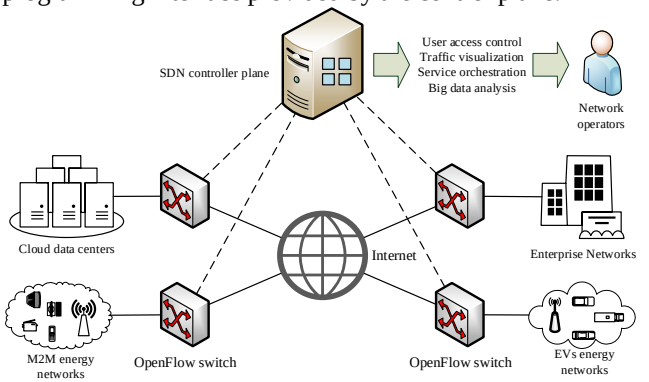


FIGURE 1. A typical SD-WAN deployment scenario.

In the above SD-WAN scenario, OpenFlow switches simply forward packets in terms of their internal flow tables installed by the controller plane. As for an arrived packet, an OpenFlow switch parses it to compute its flow identifier and match against flow tables to locate an entry. If an entry is successfully matched, its actions will be applied to the packet, generally forwarded to the next station. Otherwise, the packet is supposed to belong to an emerging flow, and will be delivered as a flow setup request to the control plane for instructions. The controller cluster generates the respective flow rule based on the global network view, and installs it to the switch. After that, the switch will process all packets within the flow in terms of the rule. In summary, flow tables are essential components of OpenFlow switches, and their storage and lookups have a significant impact on packet forwarding performance in the SD-WAN paradigm.

B. DIFFERENTIATED STORAGE ARCHITECTURE

With the evolution of OpenFlow protocol versions, flow tables are prone to expand beyond TCAM capacity especially in SD-WAN deployments. Therefore, it is inevitable to combine TCAM with other storage media such as SRAM and DRAM, to accommodate large-scale flow tables in OpenFlow switches. Each storage medium holds its own properties typically in terms of accessing speeds, addressing mode, storage capacity and price. TCAM supports wildcarding and outputs each lookup result in one clock cycle thanks to its capacity of parallel lookups on the entire data set. Thus it is a most prevailing storage medium for flow tables specified by OpenFlow. However, TCAM has limited capacity owing to its exorbitant price, high energy consumption and low integration, and it is difficult to accommodate a large number of flow entries. Different from TCAM, SRAM can be randomly accessed at fast speeds with a given address, and supports fast lookups on exact flow tables typically through hashing. Compared with SRAM, DRAM has moderate accessing speeds, low cost, and high capacity, which is suitable for accommodating a large amount of content data in abundant flow entries.

According to network traffic locality, packet traffic is not uniformly distributed over flows, but shows apparent biases between flows in both the long term and the short term. Packet flows can be distinguished into major ones with more packets and minor ones with less packets. Subsequently, we can store major and minor flows respectively in TCAM and SRAM for fast lookups. Meanwhile, all fields in a flow entry can be classified into two types: the match fields for identifying the flow, and the content fields for recording flow information. The match fields need to be matched for each arrived packet, while the content fields are accessed only for successful flow entry match. Thus we can separate the content fields from the flow entry, and independently accommodate them in DRAM. With these design principles, we build a differentiated storage architecture for large-scale flow-tables in Fig.2.

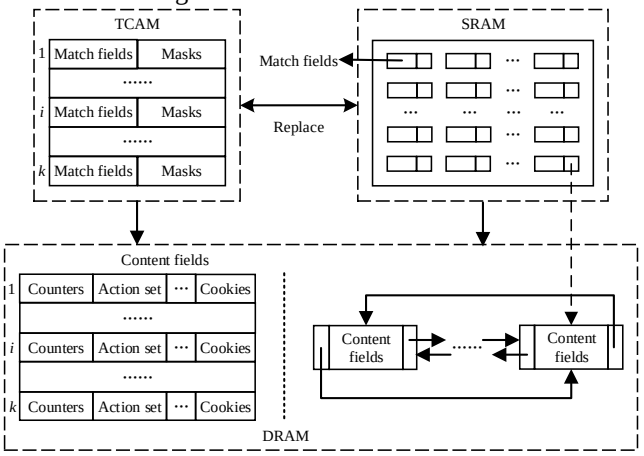


FIGURE 2. Differentiated storage architecture for large-scale flow tables.

As demonstrated in Fig.2, the match fields and the content fields of each major flow are separately kept in the TCAM and DRAM, and are associated with each other by index number or offset address. A successful lookup of the TCAM returns the offset address of the matched entry, which can be easily translated into an index number of the TCAM. Then, the index number can be utilized to directly locate the corresponding entry of sequentially organized flow entries in the DRAM. In contrast, the match fields and the content fields of each minor flow are separately maintained in the SRAM and DRAM, and are associated with each other by an additional pointer field. In particular, the pointer field is attached to the match fields of each entry in the SRAM. The flow entries in the SRAM are typically classified by their masks, and all flow entries with identical mask can be organized and searched by hashing. Once a flow entry is found, we can locate its corresponding content fields in the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2942208, IEEE Access

DRAM by its pointer field. Furthermore, all content fields of minor flows are stored as a chain structure for flexible operations, such as insertions and deletions.

In the above architecture, the major flows and the minor flows will dynamically change with the continuous accessing of flow tables by network packets arrived at the OpenFlow switch. When a new flow emerges, we treat it as a minor flow, and put its match fields and content fields respectively into the SRAM and DRAM. If it turns into a major flow with many packets arrived later, we will transfer its match fields from the SRAM into the TCAM, and its content fields from the chained list into the respective position of the sequential list in the DRAM, if there is available space in the TCAM. Otherwise, we first need to transfer the least recently accessed entry from the TCAM into the SRAM, and its content fields from the sequential list into the chained one in the DRAM. Meanwhile, the timeout mechanism is applied on both major and minor flow tables to clear out all expired flow entries in time.

The above storage architecture separates the content fields from flow entry and stores them in the DRAM with high capacity, which effectively settles the storage space pressures of TCAM even combined with SRAM. Meanwhile, we utilize TCAM and SRAM respectively to keep the match fields of major flows and minor ones. This will bring two consequences to flow table lookup performance: (a) direct hits in the TCAM for a large number of packets within major flows; (b) slow lookups in the SRAM for a small number of packets within minor flows. On average, it is expected to achieve satisfactory lookup performance of large-scale flow-tables in OpenFlow switches.

IV. MAJOR/MINOR FLOW DIFFERENTIATION

This section investigates into the traditional elephant/mice flow differentiation method, takes an insight into the packet-in-batch property of a flow, and presents the active/idle flow differentiation method to achieve higher TCAM hit rates.

A. Traditional Elephant/Mice Flow Differentiation

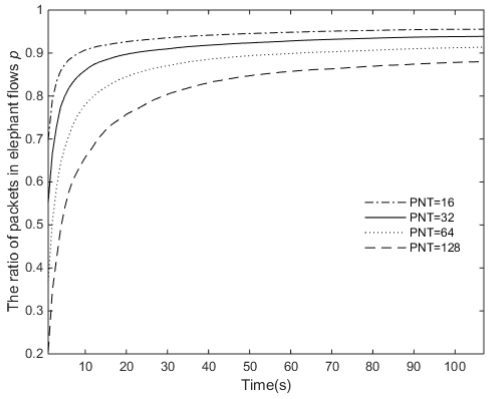
Extensive research efforts have identified the presence of locality phenomenon in packet switching networks [33][34]. Network traffic locality is primarily reflected on packet traffic distribution over flows. In particular, a majority of packets tend to concentrate on a small number of flows, while a large number of flows just account for a minority of packets. Hence packet flows could be naturally distinguished into elephant ones transferring a large number of packets and mice ones transferring a small number of packets. This is a prevalent major/minor flow differentiation method applied in traditional flow table storage architectures. The differentiation method will get a low average search time, since a majority of packets can quickly locate their flow entries in the TCAM supporting wildcarding, while only a minority of packets need to find their flow entries in the SRAM with exact matching. TCAM hit rate is a key metric to characterize the performance of major/minor flow differentiation method in the flow table storage architecture. Then we formulate the TCAM hit rate of the above Elephant/Mice Flow (EMF) differentiation method based on network traffic locality as follows.

Suppose that an OpenFlow switch transfers N packet flows, where each one transmits n packets on average. According to network traffic locality, a small number of elephant flows carry a majority of packets, and a large number of mice flow only transfer a minority of packets. Then we can assume two factors p and q respectively as the ratio of the number of packets within elephant flows to the total number of transferred packets, and that of the number of elephant flows to the total number of flows. Hence we can compute the number of elephant flows as qN, and the number of packets with elephant flows as pNn. As for the existing EMF differentiation method, each flow is initially held in the SRAM, and is transferred into the TCAM if it is identified as an elephant flow with e arrived packets. This implies that TCAM will be successfully enquired only for packets after e ones in each elephant flow. Then we compute the number of packets with successful TCAM query by subtracting the total number of packets within elephant flows pNn to that of their front packets before being transferred into the TCAM qNe. Hence, we can formulate the TCAM hit rate of the traditional EMF differentiation in the flow table storage architecture in (1).

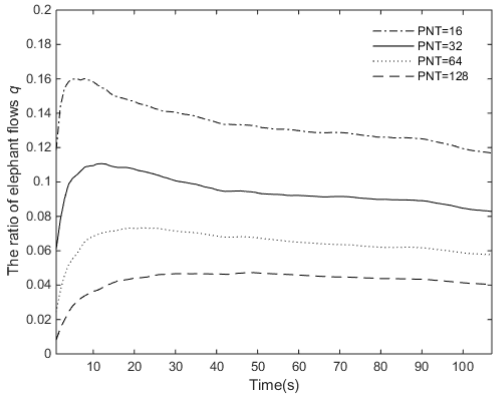
$$HR_{EMF} = \frac{pNn - qNe}{Nn} = p - \frac{qe}{n}. \quad (1)$$

As shown in (1), the TCAM hit rate chiefly depends on the two factors of network traffic locality p and q, the number of packets per flow n, and the number of packets for elephant flow identification e. Considering the ratio of e to n as the identification efficiency of elephant flows e', we can further simplify the TCAM hit rate in (1) as p-qe'. To estimate the TCAM hit rate, we apply a backbone network traffic trace TRACE20110418 [36] to measure the above parameters. The trace contains 15,420,235 packets, with duration time 107s roughly, and more details in Section IV.A. The above parameters significantly depend on the differentiation indicator of elephant/mice flows, i.e., packet number threshold (*PNT*). By setting the *PNT* respectively as 32, 64, and 128, we read packets from the traffic trace, and count the number of packets, flows, elephant flows and packets in elephant flows per second. With these numbers, we compute the above parameters p, q, e', and estimated TCAM hit rates in terms of (1) in Fig.3.

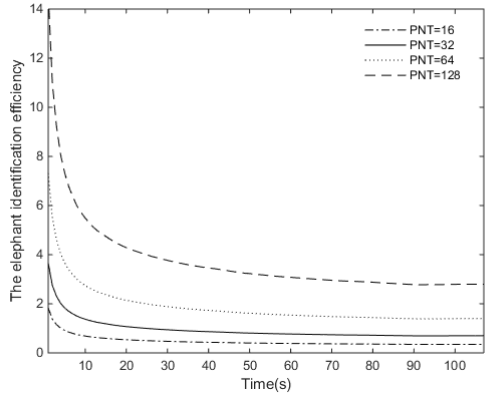
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2942208, IEEE Access



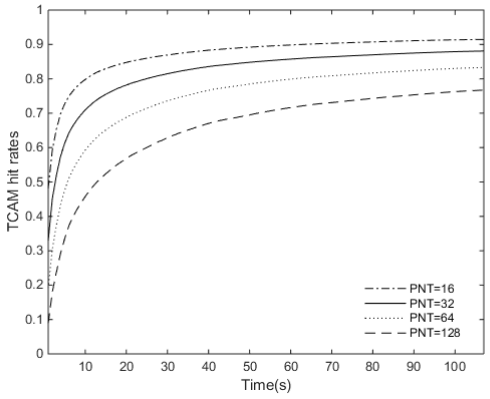
(a) the parameter *p*



(b) the parameter *q*



(c) the parameter *e'*



(d) estimated TCAM hit rates

FIGURE 3. Estimated TCAM hit rates of the EMF differentiation method and its parameter measurements.

As seen from Fig.3(a), the number of packets in elephant flows accounts for almost above 80% of total packets at stable states, even if the *PNT* is set as up to 128. In contrast, the number of these elephant flows just takes up of below 15% of total flows shown in Fig.3(b). This indicates that network traffic exhibits a high spatial locality. As shown in Fig.3(c), the identification efficiency of elephant flows e' is proportional to the number of packets for elephant flow identification *e*, i.e., the *PNT*. This is attributed to the invariance of the number of packets per flow *n*. Finally, we can see from Fig.3(d) that, estimated TCAM hit rates achieve higher than 80% at stable states based on the measurement results of the 3 parameters in Fig.3(a)-(c), if the *PNT* is set as below 64.

The above EMF differentiation method gets high TCAM hit rates for the flow table storage architecture in virtue of long-term distribution characteristics of packet traffic over flows. However, it cannot adapt to dynamical variation of packet traffic for various network application scenarios. For example, there probably is a very few or even no packet within an elephant flow in the TCAM during certain periods, which will result in inadequate utilization of the TCAM. What is worse, the differentiation method will be sharply degraded for packet traffic with heavy-tailed distribution commonly seen in various network scenarios. Therefore, it is necessary to devise a better flow differentiation method for more stable and higher TCAM hit rates in the flow table storage architecture.

B. Packet-in-batch Property

Network traffic measurements indicate that packets in a flow arrive not uniformly but intensively in batches [34][35]. This phenomenon is a natural artifact of the protocols and applications used for network data transmission. Firstly, the most widespread Internet service, i.e. World Wide Web, is generally manifested as file downloading behaviors from a network perspective. Secondly, various increasing streaming applications, such as Internet television and live streaming,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2942208, IEEE Access

generate persistent bulk data transfer activities. Thirdly, some emerging cloud storage-based services also produce a mass of file downloading/uploading activities. In summary, those activities trigger bursts of packets within particular flows, and give rise to the property of packets in batches.

The above packet-in-batch property is commonly regarded as a dynamic property of network traffic locality in terms of a packet flow. The SDN paradigm does not produce a change to the distribution of packet traffic over flows. However, it brings a more obvious packet-in-batch property for a flow, since it supports flexible flow definition by introducing wildcards into the match fields according to the OpenFlow specifications. In particular, wildcards will aggregate a set of traditional exact flows into a wildcarding flow. The aggregation is inclined to merge scattered packets in multiple exact flows into packet batches in the wildcarding flow, and small batches in multiple exact flows into large batches in the wildcarding flows. Fig.4 demonstrates a typical example of packet-in-batch property during the flow aggregation process.

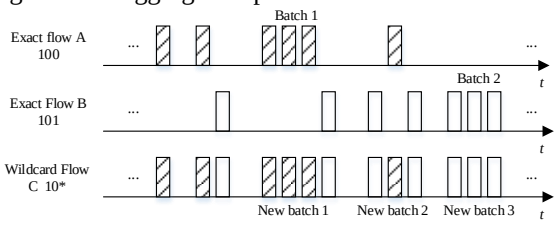
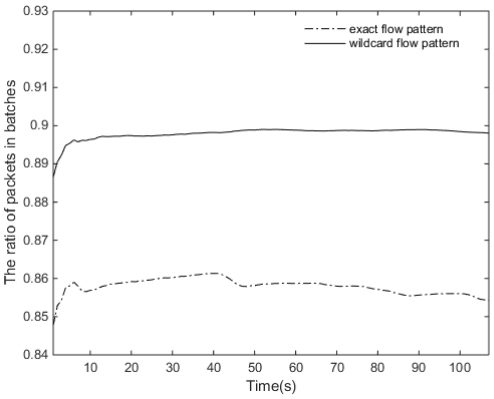


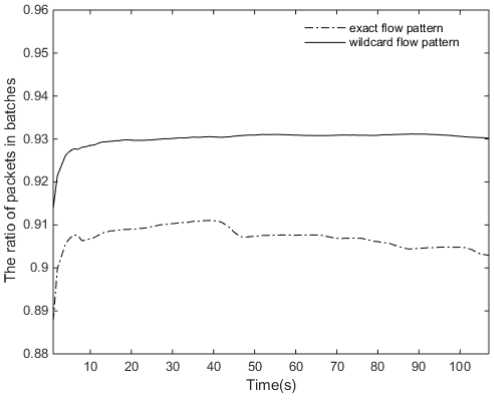
FIGURE 4. Packet-in-batch property during flow aggregation process.

As shown in Fig.4, there are two exact flows A and B with the match fields respectively valuing 100 and 101. During the observation period, both the two exact flows contain a batch of packets and a few scattered ones. We define a wildcarding flow C with the match fields valuing 10*, aggregated from the two exact flows A and B. As seen from Fig.4, the wildcarding flow C contains 3 packet batches and 3 scattered packets. This is to say, packets in the wildcarding flow C are much more intensive with more packet batches and less scattered packets than those in the exact flows A and B. In conclusion, the packet-in-batch property of a flow would be strengthened by wildcards in the match fields of flow entries.

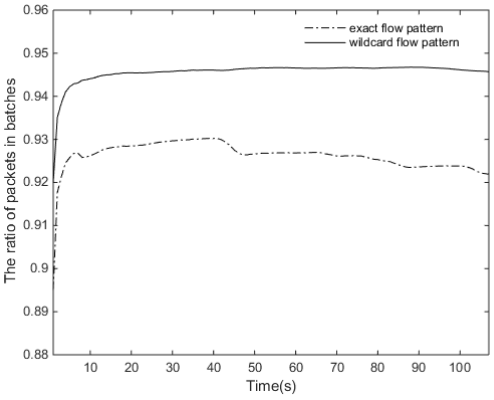
To verify the impact of wildcards on the packet-in-batch property of packet traffic in terms of flows, we devise some quantitative indicators of the property and measure them in virtue of real network traffic traces. As seen from Fig.4, the packet-in-batch property can be essentially characterized by the number of packets in batches, the number of batches, and the number of packets per batch. These quantitative indicators have great dependence on the packet interval threshold (*PIT*), which determines two adjacent packets within a flow belong to a batch or not. By setting the *PIT* respectively as 0.25s, 0.5s, 1s, and 2s, we similarly read packets from the traffic trace TRACE20110418 [36], and count the number of packets, batches, packets in batches. With these numbers, we compute the parameters, i.e., the ratio of the number of packets in batches to that of all packets, the number of batches per second, and the number of packets per batch respectively in Fig.5, Fig.6, and Fig.7.



(a) *PIT*=0.25s

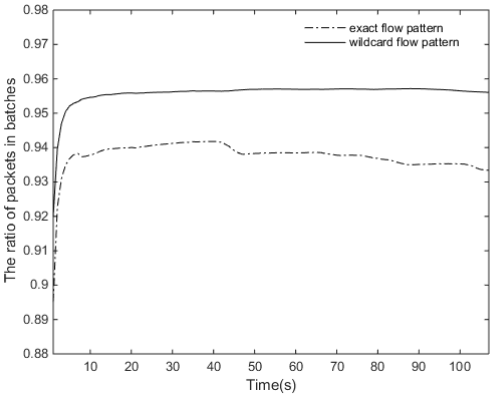


(b) *PIT*=0.5s



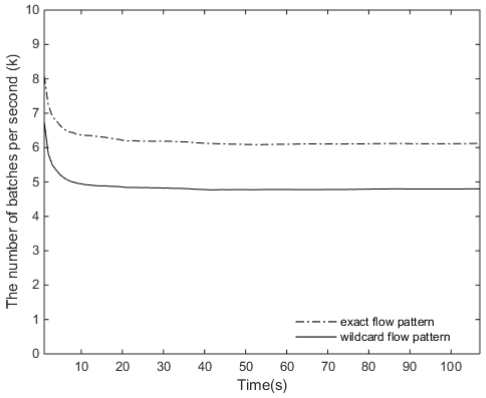
(c) *PIT*=1s

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2942208, IEEE Access

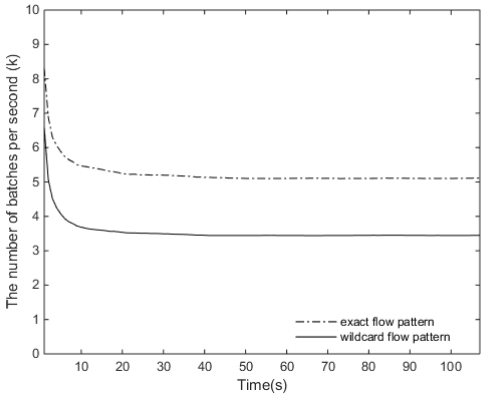


(d) *PIT*=2s

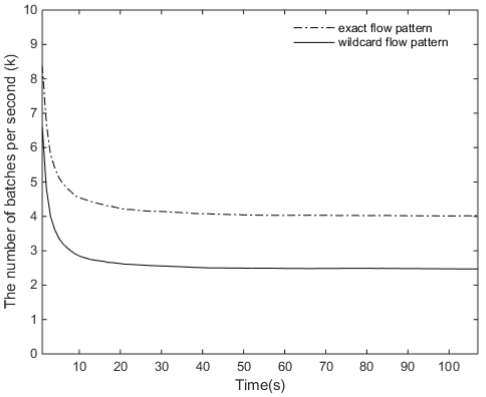
FIGURE 5. The ratio of the number of packets in batches.



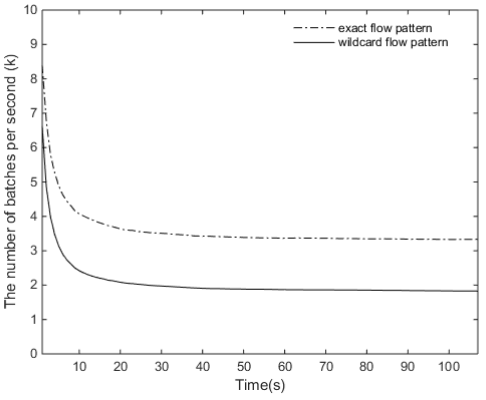
(a) *PIT*=0.25s



(b) *PIT*=0.5s

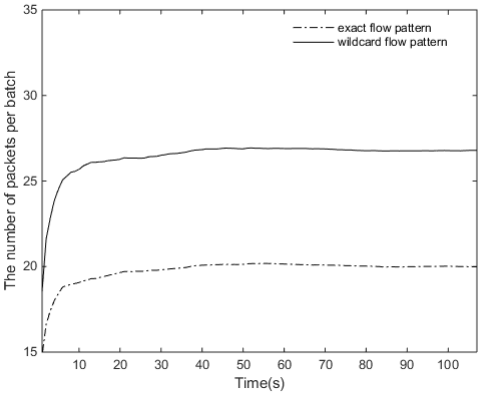


(c) *PIT*=1s



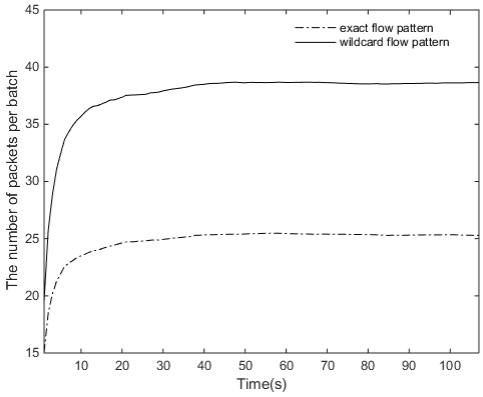
(d) *PIT*=2s

FIGURE 6. The number of batches per second.

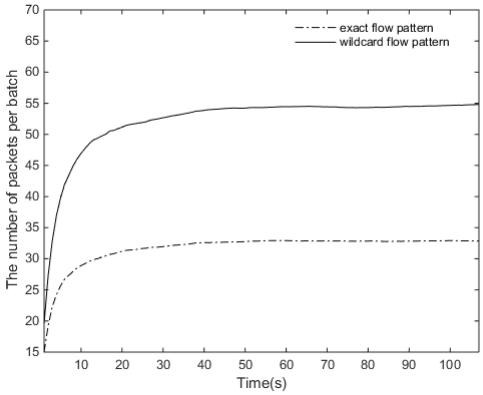


(a) *PIT*=0.25s

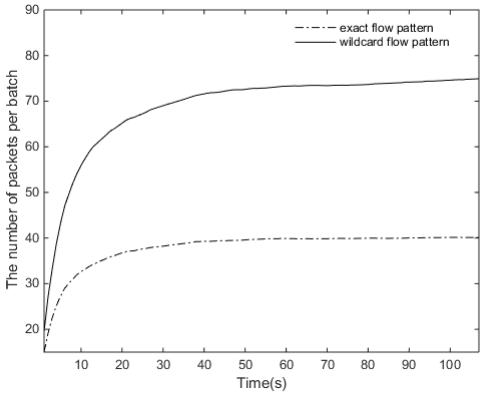
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2942208, IEEE Access



(b) *PIT*=0.5s



(c) *PIT*=1s



(d) *PIT*=2s

FIGURE 7. The number of packets per batch.

As shown in Fig.5, there are more number of packets in batches under the wildcard flow pattern than that under the exact flow pattern, no matter what the *PIT* values. This implies that a part of scattered packets under the exact flow pattern will be combined into batches under the wildcard flow pattern. Meanwhile, we can see from Fig.6 that, less number of batches arises for each second under the wildcard flow pattern than that under the exact flow pattern, regardless of the values of the *PIT*. This means that multiple batches under the exact flow pattern are merged into a single batch under the wildcard flow pattern. Consequently, there will be much more number of packets per batch under the wildcard flow pattern in Fig.7. In summary, we can conclude that the packet-in-batch property of network traffic is much more obvious under the wildcard flow pattern than that under the exact flow pattern.

C. Active/Idle Flow Differentiation

According to the above packet-in-batch property, packets in a flow will arrive at an OpenFlow switch in batches. From the viewpoint of a switch, a flow will exhibit two states: (a) active with a batch of packets being transmitted; (b) idle with just a few scattered packets or even none arrived in the near future. This inspires us to distinguish packet flows into active ones and idle ones in terms of their dynamic states. The state of a flow can be identified on the arrival of a packet by the arrival interval between it and the latest packet within the flow. In particular, we set a threshold for the packet arrival interval to distinguish the flow states. If the packet arrival interval goes below the threshold, the flow is considered to have come into the active state. Otherwise, the flow is supposed to stay at the idle state. Furthermore, the threshold can be measured and even dynamically adjusted to match the number of active flows with the capacity of the TCAM. The Active/Idle Flow differentiation method (AIF) is expected to achieve high hit rates for TCAM holding active flows, owing to its sufficient consideration of flow dynamics. We formulate its TCAM hit rate based on packet-in-batch arrivals as follows.

Suppose that an OpenFlow switch transfers N packet flows, where the ith one transmits n_i packets. By setting the threshold of packet arrival interval, all packets in a flow are divided into packets in batches and scattered packets. Then the packet flow is considered as a crossing sequence of packet batches and sparse packets (maybe none). Suppose there is k_i packet batches in the ith flow, the packet sequence in the flow can be mapped into a number sequence (b_{i1}, s_{i1}, …, b_{ij}, s_{ij}, …, b_{ik_i}, s_{ik_i}), where b_{ij} and s_{ij} respectively represents the number of the jth packet batch and sparse packets between the jth batch and the next one in the ith flow. As for the arrivals of sparse packets, the flow will be identified at the idle state, and be stored in the SRAM. When a packet batch comes, it is assumed that the flow is determined to stay at the active state and is transferred into the TCAM after a packets. The flow will be kept in the TCAM for the rest of packets in the batch. Consequently, we can calculate the TCAM hit rate of the AIF differentiation method in (2), where r stands for the ratio of the number of packets in batches to that of all transferred packets, and b denotes the average number of packets in a batch.

$$HR_{AIF} = \frac{\sum_{i=1}^{N}\sum_{j=1}^{k_i}\left(b_{ij}-a\right)}{\sum_{i=1}^{N}\sum_{j=1}^{k_i}\left(b_{ij}+s_{ij}\right)} = r\left(1-\frac{a}{b}\right). \qquad (2)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2942208, IEEE Access

As shown in (2), the TCAM hit rate primarily relies on the ratio of the number of packets in batches *r*, the average number of packets in a batch *b* and the number of packets for active flow identification *a*. According to the above active flow identification method, *a* should be set as 2, since it identifies the flow state with currently arrived packet and the latest one within a flow. With the measurements of the parameters *r* and *b* respectively in Fig.5 and Fig.7, we compute the estimated TCAM hit rates in terms of (2) in Fig.8.

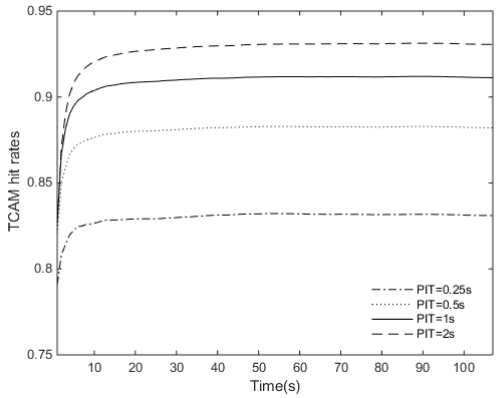


FIGURE 8. Estimated TCAM hit rates of the AIF differentiation method.

As seen from Fig.8, the estimated TCAM hit rates keep above 90% almost all the time if the *PIT* set as above 1s. By comparing Fig.8 with Fig.3(d), it can be found that the TCAM hit rates of the AIF differentiation method are higher than the EMF one in general. This can primarily attribute to the fact that, the AIF differentiation method takes advantage of short-term states of packet flows, while the EMF one makes use of long-term statistical characteristics of packet traffic. In particular, it is a common situation for elephant flows with just a few or even no packet sometimes, and tends to result in a waste of precious TCAM resources. In contrast, the AIF differentiation method places flows with a batch of transmitting packets in the TCAM, and squeeze flows without intensive packets out of the TCAM. Therefore, it has a better adaption to the dynamics of packet flows, and will achieve more sufficient utilization of the TCAM.

V. ALGORITHMIC IMPLEMENTATION

This section provides the pseudo-code implementations of packet forwarding and flow table modifications based on our proposed flow table storage architecture applying the above active/idle flow differentiation method.

A. Packet Forwarding

Our proposed flow table storage architecture is primarily manifested on packet forwarding based on flow table lookups in an OpenFlow switch. As for each arrived packet, we search flow tables to locate its flow entry, and forward it in terms of the action set in the entry. For the sake of our proposed flow table storage architecture, it needs to first search TCAM and SRAM accommodating all match fields of packet flows. If the search succeeds, we get the corresponding content fields in the DRAM by the index of the matched entry in the TCAM or the pointer field in the matched entry in the SRAM. Subsequently, we process the packet in accordance with the action set in the content fields, and update the content fields with the packet including flow state. When the flow state becomes active, the match fields if kept in the SRAM should be swapped into the TCAM. Otherwise, the switch should send a flow setup request containing the packet partly or wholly to its SDN controller, if the search on flow tables fails. Table I shows the pseudo-code implementation of packet forwarding based on our flow table storage architecture.

TABLE I. PACKET FORWARDING ALGORITHM

Algorithm 1: The forwarding algorithm of an arrival packet *p* based on our proposed flow table storage architecture in an OpenFlow switch.

```
PacketForward(Packet p)
1.  key_fields ← ParsePacket(p);
2.  mf ← GetMatchFields(key_fields, p);
3.  index ← QueryTcam(mf);
4.  if index is valid, then
5.      cf ← ReadDramSeqList(index);
6.  else
7.      pse ← QuerySram(mf);
8.      if pse is not NULL, then
9.          pcf ← pse->pointer;
10.         cf ← ReadDramLinkList(pcf);
11.     else
12.         m ← CreatePacketInMessage(p);
13.         SendMessageToController(m);
14.         return false
15.     end if
16. end if
17. ExecuteActions(cf.actions);
18. UpdateFlow(cf);
19. if mf is matched in the SRAM and p.time-cf.last_pkt_time < PIT, then
20.     if TCAM is full, then
21.         index, mf_old ← GetTcamEntryLRU();
22.         cf_old ← GetDramSeqList(index);
23.         pcf_old ← InsertDramLinkList(cf_old);
24.         InsertSramEntry(mf_old, pcf_old);
25.     else
26.         index ← GetTcamEntryEmpty();
27.     end if
28.     DeleteSramEntry(pse);
29.     InsertTcamEntry(mf, index);
30.     DeleteDramLinkList(pcf);
31.     InsertDramSeqList(cf, index);
32. end if
```

Upon receiving a packet *p*, the switch parses it and extracts its key fields from its protocol header of each layer, such as source MAC address, destination MAC address, source IP address, destination IP address, source port, destination port, protocol and etc (*line* 1). Then the match fields *mf* is gotten from these key fields and other information of the packet *p*, and is utilized to search the TCAM for a match (*line* 2-3). If the search succeeds, we will get the index of a matched entry, and obtain the content fields *cf* from the sequential list in the DRAM by the index (*line* 4-5). Otherwise, we need to further look up the SRAM for a match (*line* 6-7). If the lookup succeeds, we can

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2942208, IEEE Access

acquire the respective content fields *cf* from the DRAM by the pointer in the matched entry in the SRAM (*line* 8-10). As for the case that there is no match in both the TCAM and the SRAM, the switch will send a packet-in message containing the information of the packet *p* to its SDN controller for instructions (*line* 11-13).

With the content fields *cf*, the switch applies the actions in *cf* to the packet *p* typically forwarding it to the next station, and update them with the packet *p* such as counters and the arrival time of the latest packet (*line* 17-18). If the match fields *mf* is successfully matched in the SRAM, we should determine whether the flow turn into the active state, by comparing the interval between the arrival time of the packet *p* and that of the latest packet in *cf* with the packet interval threshold *PIT* set in the switch (*line* 19). If the flow becomes active, its match fields and content fields are respectively transferred from the SRAM into the TCAM and from the link list to the sequential list in the DRAM (*line* 28-31). Before the transfer, it is necessary to find a vacant entry in the TCAM (*line* 26). If the TCAM is full, it still needs to find the flow in the TCAM without any arrived packet for a long time (*line* 20-21). Then we shift its match fields and content fields respectively from the TCAM into the SRAM and from the sequential list to the link list in the DRAM (*line* 22-24).

B. Flow Table Modifications

In addition to the above lookups and updates during packet forwarding, there are other operations on flow tables, such as insertions and deletions. Tables II illustrates the pseudo-code implementation of flow table insertion in an OpenFlow switch on the arrival of a flow modification message from its SDN controller. Upon receiving a FLOW_MOD message with the ADD command, the switch will extract the match fields from it and create the content fields with it (*line* 1-3). If the TCAM is not full, we find an empty entry with the position *index*, and insert the match fields and the content fields respectively at the position of the SRAM and the sequential list in the DRAM (*line* 4-7). Otherwise, we insert the content fields into the linked list in the DRAM, and the match fields into the SRAM (*line* 9-10).

TABLE II. FLOW TABLE INSERTION ALGORITHM

Algorithm 2: The insertion into flow tables in an OpenFlow switch on the arrival of a flow modification message *m* from its controller.

```
Insert(Message m)
1. if m.type==FLOW_MOD and m.command==ADD, then
2.    mf ← GetMatchFields(m);
3.    cf ← CreateContentFields(m);
4.    if TCAM is not full, then
5.       index ← GetTcamEntryEmpty();
6.       InsertTcamEntry(mf, index);
7.       InsertDramSeqList(cf, index);
8.    else
9.       pcf ← InsertDramLinkList(cf);
10.      InsertSramEntry(mf, pcf);
11.   end if
12.end if
```

Tables III exhibits the pseudo-code implementation of flow table deletion in an OpenFlow switch on the arrival of a flow modification message from its SDN controller. As for a FLOW_MOD message with the DELETE command, the switch extracts the match fields from it (*line* 1-2) and matches them against flow tables in the TCAM and the SRAM (*line* 3, 8). If the match succeeds to get a position *index* in the TCAM, we reset the entry in the TCAM and that of the sequential list in the DRAM at the position (*line* 4-6). Otherwise, if the match succeeds to return an entry pointer *pse* in the SRAM, we delete the entry in the linked list in the DRAM by its pointer *pse->pcf*, and the entry in the SRAM by its pointer *pse* (*line* 9-11). We should return false if the match fails in both the TCAM and the SRAM.

TABLE III. FLOW TABLE DELETION ALGORITHM

Algorithm 3: The deletion on flow tables in an OpenFlow switch on the arrival of a flow modification message *m* from its controller.

```
Delete(Message m)
1. if m.type==FLOW_MOD and m.command==DELETE), then
2.    mf ← GetMatchFields(m);
3.    index ← QueryTcam(mf);
4.    if index is valid, then
5.       ResetTcamEntry(index);
6.       ResetDramSeqList(index);
7.    else
8.       pse ← QuerySram(mf);
9.       if pse is not NULL, then
10.         DeleteDramLinkList(pse->pcf);
11.         DeleteSramEntry(pse);
12.      else
13.         return false;
14.   end if
15.end if
```

Tables IV shows the pseudo-code implementation of flow table timeout scanning in an OpenFlow switch. As for each active flow, we get its content fields containing its two timeout fields *idle_timeout* and *hard_timeout* from the sequential list in the DRAM (*line* 1-2). If either of them expire, we will reset its entry of match fields in the TCAM, and its entry of content fields in the sequential list in the DRAM (*line* 3-5). Similarly for each idle flow, we get its two timeout fields from the linked list in the DRAM, and determine whether either of them expire (*line* 8-9). As for each expired idle flow, we will delete its entry of match fields in the SRAM, and its entry of content fields in the linked list in the DRAM (*line* 10-11).

TABLE IV. FLOW TABLE TIMEOUT SCANNING ALGORITHM

Algorithm 4: The timeout of flow tables in an OpenFlow switch.

```
Timeout(time t)
1. for each position i of the sequential list in the DRAM, do
2.    cf ←GetDramSeqList(i);
3.    if (t - cf.last_pkt_time >= cf.idle_timeout || t - cf.setup_time >=
cf.hard_timeout), then
4.       ResetTcamEntry(i);
5.       ResetDramSeqList(i);
6.    end if
7. end for
8. for each entry with the pointer p in the linked list in the DRAM, do
9.    if (t – p->last_pkt_timeout >= p->idle_time || t - p->setup_time >=
p->hard_timeout), then
```

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2942208, IEEE Access


```
10.      DeleteSRAMEntry(p->pse);
11.      DeleteDramLinkList(p);
12.   end if
13.end for
```

VI. EXPERIMENTS

This section introduces our experimental methodology, and evaluates the performance of our proposed flow table storage architecture in terms of TCAM hit rates and average flow table access time.

A. Experimental Methodology

For convenient evaluation and comparison, we implement different flow table storage architecture by simulations with C/C++ programming. The simulation program will be repeatedly executed to perform packet forwarding off-line on real network traffic traces. In particular, it reads packets from each traffic trace one by one, parses them to get their key fields from the protocol header at each layer, and obtain their match fields for matching against flow tables. After that, it needs to search the flow tables separately keeping major flows and minor ones for a matched entry. If the search succeeds, we update the content fields of the matched entry with the packet such as flow states. When a minor flow is distinguished to become a major one for a packet, its entry should be transferred from the minor flow table into the major one. However, if the search fails, we will directly create a new flow entry with the packet and insert it into the minor flow table. Meanwhile, the program will write down necessary statistical information, such as the number of active/elephant flows and that of packets hitting the TCAM.

For simplicity, we select the classical 5 fields identifying a connection/session as the match fields of flow tables in our experiments. In particular, the 5 fields include protocol type, source IP, destination IP, source port, and destination port. The masks of these fields are set as follows: *0xff* for protocol type, *0xff00* for both source port and destination port, and default subnet mask for both source IP and destination IP (*0xffffffff* for classes D and E addresses). By this way, we can get 16 types of masks. Besides, we set 10*s* as the timeout of each flow entry.

In our experiments, we select two network traffic traces, TRACE20110418 and TRACE20130903 [36], collected from a 10Gps main channel at the border of Jiangsu Province in the CERNET. Each trace contains 15,420,235 packets, gathered with a ratio of 1:4. The 2 traces respectively last for 107*s*, 100*s* roughly. With the above configurations, we can provide the varying number of simultaneous flows for the two traces in Fig.9. As seen from Fig.9, both traces contain a large number of simultaneous flows up to between 60k and 70k most of the time.

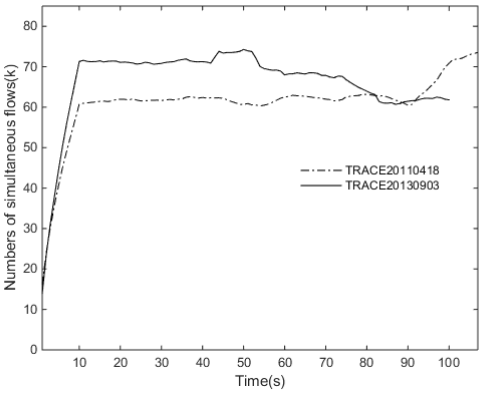
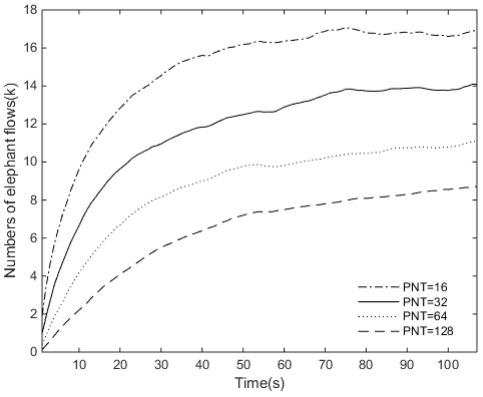


FIGURE 9. The number of simultaneous flows.

B. The Stability of Major Flow Quantity

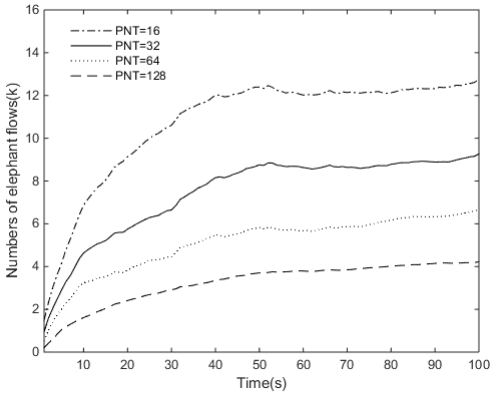
The large-scale flow table storage architecture has a strong expectation on major flows to keep a steady quantity, as their storage media TCAM has a fixed capacity once deployed. Thus the quantitative stability of major flows is a key metric to different major/minor flow differentiation method in large-scale flow table storage architecture.

As for the traditional EMF differentiation method, we get the real-time number of elephant flows for the two traffic traces by setting different values of the *PNT* in Fig.10. As shown in Fig.10, the number of elephant flows has been gradually increasing all the time regardless of the traffic traces and the values of the *PNT*. This is to say, it is hard to find a suitable value of the *PNT* to achieve a stable number of elephant flows. As a consequence, the traditional EMF differentiation method is unqualified as the major/minor one in the flow table storage architecture.



(a) TRACE20110418

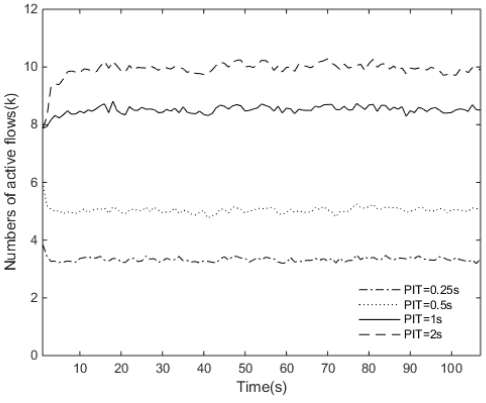
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2942208, IEEE Access



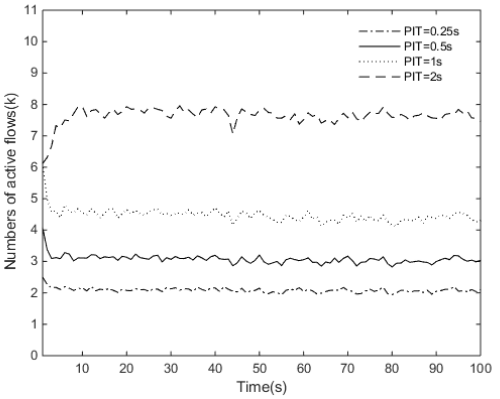
(b) TRACE20130903

FIGURE 10. The number of elephant flows.

As for our proposed AIF differentiation method, we obtain the real-time number of active flows by setting different values of the *PIT* in Fig.11. We can see from Fig.11 that, the number of active flows always keeps stable, no matter which traffic trace and which value of the *PIT*. Meanwhile, the number of active flows steadily grows with the increasing value of the *PIT*. This implies that we can match the number of active flows with the TCAM capacity, by fine-grained adjusting the value of the *PIT*. In particular, the *PIT* is suitable to be configured as 1s and 2s respectively for TRACE20110418 and TRACE20130903, if the TCAM has a capacity of accommodating 8k flows.



(a) TRACE20110418



(b) TRACE20130903

FIGURE 11. The number of active flows.

C. TCAM Hit Rate

TCAM hit rate is a key performance metric of large-scale flow table storage architecture applying different major/minor flow differentiation method. As for our proposed AIF one, the *PIT* has a significant impact on the TCAM hit rate. Suppose the TCAM has a fixed capacity of 8*k* flow entries, we implement the flow table storage architecture according to the experimental methodology in Section VI.A. By increasing the value of the *PIT*, we count the number of packets hitting the TCAM for each traffic trace, and achieve the relationship of the TCAM hit rates with the *PIT* in Fig.12. As seen from Fig.12, both traces share highly similar variation rules of the TCAM hit rates. In particular, the TCAM hit rate sharply increases to above 0.9 before the *PIT* reaches 0.1s, and tends to be stable after the *PIT* goes beyond 1s. This is to say, the *PIT* is suitable to be set as 1s for achieving high TCAM hit rates.

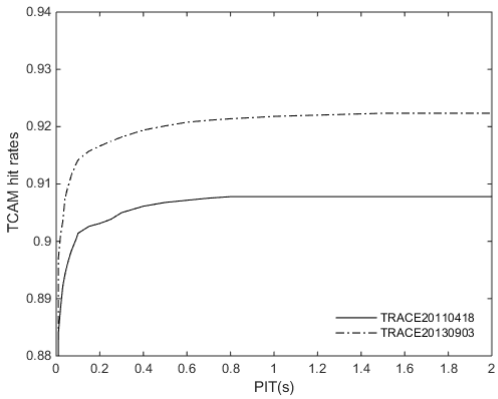
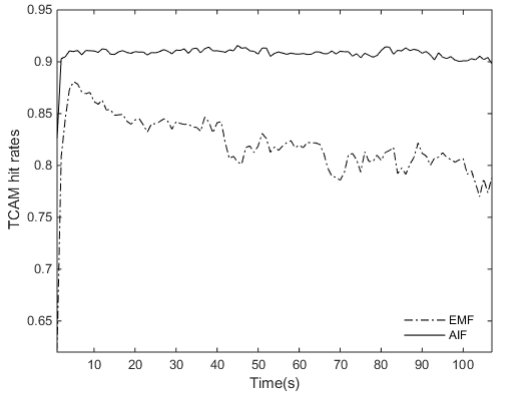


FIGURE 12. The relationship of TCAM hit rates with the *PIT*.

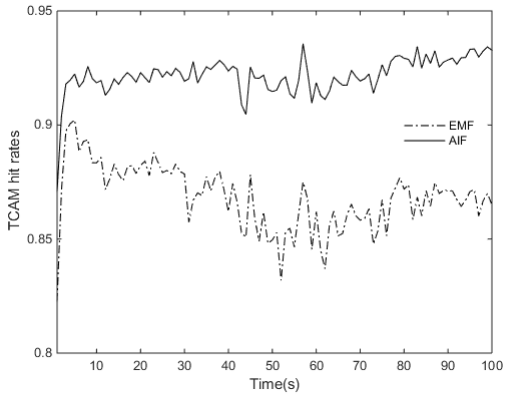
To compare between the TCAM hit rates of the flow table storage architecture applying different major/minor flow differentiation method, we set 1s for the *PIT* in our proposed AIF differentiation method, and choose the top 8*k* flows with the most number of packets as elephant flows for the EMF differentiation method. Fig.13 demonstrates the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2942208, IEEE Access

TCAM hit rates of both major/minor flow differentiation methods. As seen from Fig.13, our proposed AIF differentiation method achieves higher and more stable TCAM hit rates than the traditional EMF one. In particular, the TCAM hit rates of the AIF differentiation method keep above 0.9 for both traces almost all the time. On the contrary, the TCAM hit rates of the EMF differentiation method fluctuate between 0.8 and 0.9, and even have declining trend with time goes by.



(a) TRACE20110418



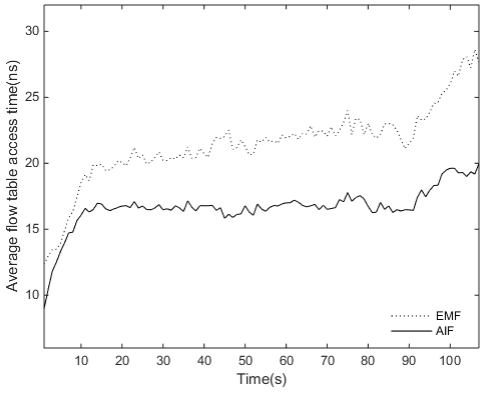
(b) TRACE20130903

FIGURE 13. The comparison of TCAM hit rates.

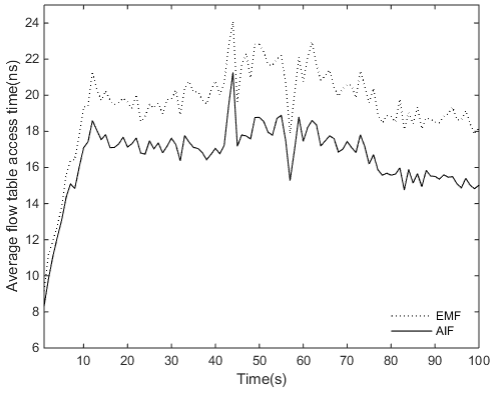
D. Average Flow Table Access Time

Our proposed AIF differentiation method achieves high TCAM hit rates by accommodating active flows currently transferring packets intensively in the TCAM. However, it will bring more flow entry replacements in the TCAM, as a flow generally switches between the active state and the idle one repeatedly. Thus, we eventually compare average flow table access time of flow table storage architecture applying different major/minor flow differentiation method. In particular, the time is primarily composed of TCAM lookup and replacement time, SRAM lookup time, and DRAM access time, during the packet forwarding process. Each type of time will be calculated by dividing the access times of the corresponding memory by its access frequency.

As for experimental configurations, we set access frequencies of the storage medium TCAM/SRAM/DRAM respectively as 450/450/200MHz, 333/333/166MHz and 200/200/133MHz. Moreover, the flow table of each mask in the SRAM is set with its hash length as 2^{10}. Then we read packets from selected traffic traces, perform packet forwarding algorithm, and count access times of each packet respectively in TCAM, SRAM and DRAM. Eventually, we compute flow table access time per packet for our proposed flow table storage architecture with different major/minor flow differentiation methods in Fig.14-Fig.16.

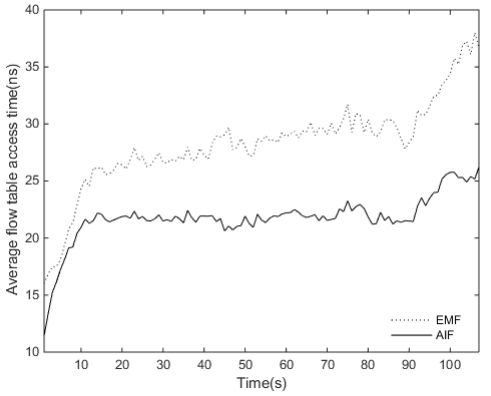


(a) TRACE20110418



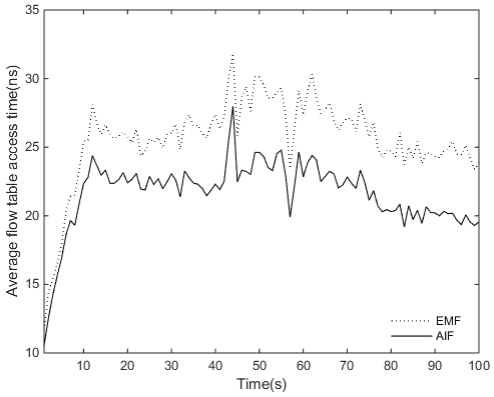
(b) TRACE20130903

FIGURE 14. The average flow table access time with TCAM450MHz, SRAM450MHz and DRAM200MHz.



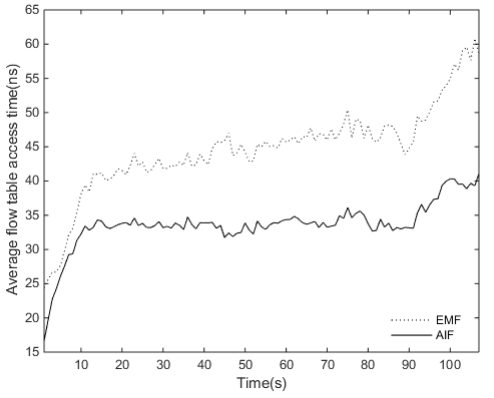
(a) TRACE20110418

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2942208, IEEE Access

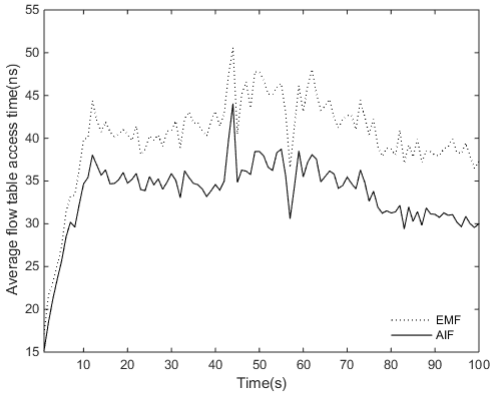


(b) TRACE20130903

FIGURE 15. The average flow table access time with TCAM333MHz, SRAM333MHz and DRAM166MHz.



(a) TRACE20110418



(b) TRACE20130903

FIGURE 16. The average flow table access time with TCAM200MHz, SRAM200MHz and DRAM133MHz.

We can see from Fig.14-Fig.16 that, our proposed AIF differentiation method has less flow table access time per packet than the traditional EMF one all the time, regardless of the traffic traces and the TCAM/SRAM/DRAM access frequencies. Meanwhile, both flow differentiation methods almost share identical variation rules of average flow table access time for different access frequencies of the storage medium TCAM/SRAM/DRAM. This infers that the relative performance of each differentiation method is independent of TCAM/SRAM/DRAM access frequencies. Moreover, TCAM replacement rates have a minor penalty to the flow table lookup performance, as they are almost negligible compared with the benefits from the increasing of TCAM hit rates.

VII. CONCLUSION

Flow tables are essential components in OpenFlow-based SDN data plane. However, they are sharply expanded, by the increasing number of fields introduced into flow entry by OpenFlow specifications, and a large number of flow entries for large-scale SDN deployments typically in wide-area networks. This paper is thus motivated to propose an efficient differentiated flow table storage architecture, which accommodates the match fields of active flows and idle ones respectively in TCAM and SRAM, and the content fields of both types of flows in DRAM. In particular, active flows and idle ones are distinguished by a threshold of packet arrival intervals based on the property of packet-in-batch arrivals within a flow specified by OpenFlow. Simulation experiments with backbone network traffic traces reveal that, our proposed storage architecture can match the number of active flows with the TCAM capacity by adjusting the threshold of packet arrival intervals, and achieve higher TCAM hit rates and lower average flow table access time than the conventional one applying the elephant/mice flow differentiation method.

REFERENCES

[1] Mckeown N, Anderson T, Balakrishnan H, et al. OpenFlow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 2008, 38(2): 69-74.

[2] Hakiri A, Gokhale A, Berthou P, et al. Software-defined networking: Challenges and research opportunities for future internet. Computer Networks, 2014, 75: 453-471.

[3] Liao Z, Zhang R, He S, et al. Deep learning-based data storage for low latency in data center networks. IEEE Access, 2019, 7(1): 26411-26417.

[4] Jain S, Kumar A, Mandal S, et al. B4: experience with a globally-deployed software defined WAN//2013 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM), Hong Kong, China, 2013: 3-14.

[5] Hong C Y, Mandal S, A F Mohammad, et al. B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in Google's software-defined WAN//2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM), Budapest, Hungary, 2018: 74-87.

[6] Open Networking Foundation. OpenFlow switch specification Version 1.0.0 [S/OL], https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf, 2009.12.31.

[7] Open Networking Foundation. OpenFlow switch specification Version 1.5.0 [S/OL], https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.0.pdf, 2014.12.19.

[8] Guo Z H, Xu Y, Liu R Y, et al. Balancing flow table occupancy and link utilization in software-defined networks. Future Generation Computer Systems, 2018, 89: 213-223.

[9] Guo Z H, Liu R Y, Xu Y, et al. STAR: Preventing flow-table overflow in software-defined networks. Computer Networks, 2017, 125: 15-25.

[10] Mimidis-Kentis A, Pilimon A, Soleret J, et al. A novel algorithm for flow-rule placement in SDN switches//4th IEEE International Conference on Network Softwarization (NetSoft), Montreal, Canada, 2018: 1-9.

[11] Mimidis A, Caba C, Soler J. Dynamic aggregation of traffic flows in SDN: Applied to backhaul networks//2nd IEEE International Conference on Network Softwarization (NetSoft), Seoul, Korea, 2016: 136-140.

[12] Luo S, Yu H, Li M. Fast incremental flow table aggregation in SDN//23rd International Conference on Computer Communication and Networks (ICCCN), Shanghai, China, 2014: 1-8.

[13] Leng B, Huang L, Wang X, et al. A mechanism for reducing flow tables in software defined network//IEEE International Conference on Communications (ICC), London, UK, 2015: 1-15.

[14] Bera S, Misra S, Jamalipour A. FlowStat: Adaptive flow-rule placement for per-flow statistics in SDN. IEEE Journal on Selected Areas in Communications, 2019, 37(3): 530-539.

[15] Kannan K, Banerjee S, Sivaraman A. Compact TCAM - flow entry compaction in TCAM for power aware SDN//Internation al Conference on Distributed Computing and Networking (ICDCN), Mumbai, India, 2013: 439-444.

[16] Guo Z H, Xu Y, Cello M, et al. JumpFlow: Reducing flow table usage in software-defined networks. Computer Networks, 2015, 92: 300-315.

[17] Banerjee S, Kannan K. Tag-In-Tag: Efficient flow table Management in SDN switches//10th IEEE International Conference on Network and Service Management (CNSM), Rio de Janeiro, Brazil, 2014: 09-117.

[18] Ge J, Chen Z, Wu Y, et al. H-SOFT: A heuristic storage space optimisation algorithm for flow table of OpenFlow. Concurrency and Computation: Practice and Experience, 2015, 27(13): 3497-3509.

[19] Naous J, Erickson D, Covington G A, et al. Implementing an OpenFlow switch on the NetFPGA platform//4th ACM/IEEE Symposium on Architecture for Networking and Communications Systems (ANCS), San Jose, USA, 2008: 1-9.

[20] Naous J, Gibb G, Bolouki S, et al. NetFPGA: Reusable router architecture for experimental research//International Conference on ACM workshop on Programmable Routers for Extensible Services of Tomorrow (PRESTO), Seattle, USA, 2008: 1-7.

[21] Matsumoto N, Hayashi M. LightFlow: Speeding up GPU-based flow switching and facilitating maintenance of flow table//13th International Conference on High Performance Switching and Routing (HPSR), Belgrade, Serbia, 2012: 76-81.

[22] Matsumoto N, Hayashi M, Morita I. GPU-accelerated hash and wildcard hybrid flow switching for tackling massive flow entries//14th IEEE International Conference on High Performance Switching and Routing (HPSR), Taipei, Taiwan, 2013: 213-214.

[23] Lee B S, Kanagavelu R, Aung K M M. An efficient flow cache algorithm with improved fairness in Software-Defined Data Center Networks//2nd IEEE International Conference on Cloud Networking (CloudNet), San Francisco, USA, 2013: 18-24.

[24] Naga K, Omid A, Jennifer R, et al. CacheFlow: Dependency-aware rule-caching for software-defined networks//ACM Symposinum on SDN Research (SOSR), 2016: 1-12.

[25] Ding X, Zhang Z, Jia Z, et al. Unified nvTCAM and sTCAM architecture for improving packet matching performance//18th ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems, Barcelona, Spain, 2017: 91-100.

[26] Wang D, Li Q, Jiang Y, et al. Balancer: A traffic-aware hybrid rule allocation scheme in software defined network//26th International Conference on Computer Communication and Networks (ICCCN), Vancouver, Canada, 2017: 1-9.

[27] Cheng Y C , Wang P C. Scalable multi-match packet classification using TCAM and SRAM. IEEE Transactions on Computers, 2016, 65(7): 2257-2269.

[28] Lee D, Wang C, Wu A. Bundle-updatable SRAM-based TCAM design for OpenFlow-compliant packet processor. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2019: 1-5. (doi: 10.1109/TVLSI.2019.2891507)

[29] Matsumoto N, Hayashi M, Morita I. LightFlow: Leveraging combination of hash and wildcard tables for high performance flow switching in large number of flow entries//10th USENIX Symposium on Networked Systems Design and Implementation (NSDI), Yokohama, Japan, 2013:1.

[30] Ferkouss O E, Snaiki I, Mounaouar O, et al. A 100gig network processor platform for Openflow//7th International Conference on Network and Service Management (CNSM), Paris, France, 2011: 1-4.

[31] Li C Q, Dong Y Q, Wu G X. OpenFlow table lookup scheme integrating multiple-cell hash table with TCAM. Journal on Communications, 2016, 37(10): 128-140.

[32] Li C Q, Dong Y Q, Wu G X, et al. A cost-effective lookup scheme combining hash table with TCAM for OpenFlow//International Conference on Network Infrastructure and Digital Content (IC-NIDC), Guiyang, China, 2018: 289-294.

[33] Duffield N, Lund C, Thorup M. Estimating flow distributions from sampled flow statistics. IEEE/ACM Transactions on Networking, 2005, 13(5):933-946.

[34] Jain R, Routhier S. Packet trains-measurements and a new model for computer network traffic. IEEE journal on selected areas in Communications, 1986, 4(6): 986-995.

[35] Klemm A, Lindemann C, Lohmann M. Modeling IP traffic using the batch Markovian arrival process. Performance Evaluation, 2003, 54(2): 149-173.

[36] Network traffic traces, http://iptas.edu.cn/src/system.php.