MixedCache: Enabling Flow Directed Rule-Caching Scheme based on Heterogeneous Cache for OpenFlow

Jialun Yang¹, *, Yi Yang¹, Jiahao Liu¹, Rulin Liu¹, Peng Qiao², and Baosheng Wang¹

¹College of Computer Science and Technology, National University of Defense Technology, Changsha, 410073, China;
²Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, Changsha, 410073, China;
yangjialun14@nudt.edu.cn

Abstract. OpenFlow switches in SDN use Multiple Flow Tables (MFTs) for fine-grained flow control. Commodity switches integrate hardware storage resources such as SRAM and TCAM to store flow tables to achieve high-speed lookups. Many increased flow tables are rapidly exhausting these hardware storage resources, which makes the switches have to balance high-speed search and massive storage. The rule-caching scheme is a popular method to solve this problem, which caches the most commonly used rules into hardware storage resources. The existing rule-caching schemes are based on single hardware storage resources, and they cannot flexibly adjust the caching strategy according to the traffic characteristics. Simultaneously, the deployed commodity switches face the problem of difficulty in changing the size of SRAM and TCAM. This paper innovatively proposes the MixedCache scheme, which makes full use of the hardware storage resources in the switch according to the skewed characteristics of network traffic. MixedCache stores the large flows in SRAM by exact match and stores the small flows in the TCAM by wildcard match. MixedCache does not need to change the size of the deployed switch hardware storage resources, but makes full use of existing resources. Compared with the rule-caching scheme based on the exact match, the cache hit rate can increase by up to 15.61%. Compared with the rule-caching scheme based on the wildcard match, the cache hit rate can increase by up to 29.69%.

1. Introduction

OpenFlow is a representative technology in SDN. In OpenFlow switch, match-action rules are stored in the flow tables in the form of entries. OpenFlow rules as flow table entries describe how OpenFlow switch process packets [1, 2]. Specifically, an OpenFlow rule can decide where the packet will be transferred or drop the packet according to the packet L1, L2, L3, and L4 headers [3–5]. To achieve more flexible flow control, OpenFlow introduces the Multiple Flow Tables (MFTs). In OpenFlow 1.1, MFTs are organized as a pipeline and process the incoming packets successively.

Commodity switches integrate hardware storage resources such as Static Random-Access Memory (SRAM) [6] and Ternary Content Addressable Memory (TCAM) [7 8] to store flow tables to achieve high-speed lookups. The size of the hardware storage resources is hard to change after the deployment of the switch. However, the number of OpenFlow rules continues to increase, even exceeding hardware storage resources [9]. The rule-caching scheme is the most commonly used method to solve this problem.
It caches the most commonly used rules into hardware storage resources. The rest of the rules are stored in the software.

The existing rule-caching schemes can be divided into two categories. One is the exact-match scheme, and the other is the wildcard-match scheme. Due to the simplicity of implementation, many products have adopted the exact-match scheme like the first version of Open vSwitch (OVS) [10]. The wildcard-match scheme can maintain the wildcard rules, which makes the best use of TCAM resources. SRAM can only be used for the exact match, TCAM can be used for both exact match and wildcard match. However, TCAM is power-hungry, expensive, and takes up more silicon space than SRAM. In commodity switches, the size of TCAM is much smaller than the SRAM [6, 8].

Based on the observations that in real-world network traffic, the flow distribution is skewed [11]. We can use the different flow characteristics of large flow and small flow. In the context of MFTs, we propose the MixedCache scheme, which makes the most use of the SRAM and TCAM in the switch. MixedCache caches the large flows in SRAM by exact match, which only caches the large flows’ hash results. The number of large flows only takes a small part of all flows, which avoids frequent updates in SRAM flow tables. A wildcard rule can define actions for multiple flows. The rules of the small flows are cached in the TCAM by wildcard match. The wildcard rule-caching scheme used in MixedCache is our previous work PipeCache [12].

Our contributions include an innovative rule-caching scheme that combines the exact match and the wildcard match. MixedCache has achieved the highest cache hit rates under different cache sizes and traffic locality conditions compared with other rule-caching schemes based on MFTs. We organize the rest of this paper as follows. In Section 2, we introduce the background of our work. In Section 3, we present our rule-caching scheme MixedCache. The experiment results are shown in Section 4. Section 5 is the conclusion of this paper and our future work.

2. Background

In OpenFlow 1.0, packet forwarding within an OpenFlow switch is controlled by a single flow table that contains a set of rules installed by the controller [2]. The OF1.1 introduces the MFTs in OpenFlow switches. Furthermore, it supports 12 match fields. The newly OF1.3 even supports 42 match fields, which significantly increases the OpenFlow flow control flexibility. This new structure brings new challenges to the OpenFlow matching methods [13].

Except for OVS’s microflow cache introduced in section 1, Microsoft’s virtual switch platform Virtual Filtering Platform VFP [14,15] shares the same idea of OVS [10]. The size of the cache and the flow distribution of traffic decide the exact-match algorithm’s performance instead of the number of flow tables in the OpenFlow pipeline. The fact that the first packet of each flow must go through the OpenFlow pipeline is inevitable, which causes the exact-match scheme’s poor performance when the flows often change [11].

Because TCAM supports fast wildcard lookups, many rule-caching schemes choose to cache the most popular or important rules in the TCAM of OpenFlow switches as cache entries. This is an effective way to avoid the problem of insufficient TCAM sizes. The rule dependency problem is an inevitable obstacle to the wildcard-match scheme [16-21].

There are multiple hardware storage resources in modern commodity switches, such as high-speed TCAM that supports the wildcard match, SRAM that only supports the exact match [22,23]. Naous designed an OpenFlow switch that combines the on-chip TCAM and the off-chip SRAM. The final lookup result is determined by the arbiter, thereby reaching the flow table line-speed lookup [24, 25]. Li Chunqiang used TCAM to store the flow table entries with hash collisions in the SRAM flow table [26]. These schemes have not considered the influence of network traffic characteristics on the performance of flow table lookup, and they cannot be applied to MFTs.

To better utilize the advantages of the two matching methods of exact match and wildcard match and maximize the use of all hardware resources in the switch, we further propose a hybrid rule-caching scheme called MixedCache, which mixes exact match and wildcard match. MixedCache selects the best matching path for incoming packets according to the characteristics of the traffic. Moreover, it can
preserve the correctness of forwarding and maximize the cache hit rate to improve the switch packet processing performance.

As shown in Table 1, compared with MixedCache, the existing rule-caching scheme either has poor performance or cannot be directly applied in the MFTs and cannot fully utilize all hardware storage resources.

| Scheme          | Matching Method | Support MFTs | Hardware | Traffic Requirement | Difficulty of Implementation | Performance |
|-----------------|-----------------|--------------|----------|---------------------|------------------------------|-------------|
| Microflow cache [10, 14] | Exact          | Yes          | N/A      | Low entropy         | Easy                         | Low         |
| [16-20]         | Wildcard        | No           | TCAM     | N/A                 | Medium                       | High        |
| PipeCache [12]  | Wildcard        | Yes          | TCAM     | N/A                 | Medium                       | High        |
| [24-26]         | Both            | No           | SRAM &   | N/A                 | Medium                       | High        |
| MixedCache      | Both            | Yes          | SRAM &   | N/A                 | Medium                       | High        |

3. The Design and Algorithms

3.1. Problem Statement

In the context of the widespread use of MFTs, to make full use of the SRAM and TCAM resources in commodity switches, we work on making fair use of the advantages of the two matching methods of exact match and wildcard match.

In our MixedCache algorithm, the input is the incoming packets. For each incoming packet, the algorithm chooses the best matching method for the packet. Moreover, the algorithm updates the cache tables according to the match results of the packet. Our algorithm's output is the entries stored in cache tables and the actions of the incoming packets. Our goal is to design a suitable cache structure for the MFTs in switches that combine the SRAM and the TCAM and maximize the traffic that hits the cache tables.

3.2. MixedCache Structure

Figure 1 shows the structure of MixedCache that consists of the MFTs in software, cache, and Cache manager in hardware. There are two kinds of hardware storage resources, TCAM and SRAM, in the cache. The function of TCAM and SRAM is to cache the selected rules in wildcard match and exact match, respectively. The cache manager is responsible for selecting the best matching path for the incoming packets.

In our design, when a packet comes, the cache manager decides whether the packet belongs to a large flow or a small flow. The packets of the small flows are sent to the TCAM for the wildcard match.
Because several small flows that arrive in a short time usually have the same operation, a single rule can define their actions. The packets of the large flows are sent to the SRAM for the exact match. The number of large flows accounts for a small part of the total number of flows. Therefore, the exact matching entries stored in the SRAM will not frequently change, reducing the controller's overhead. We use PipeCache as the rule-caching scheme in the TCAM. The PipeCache scheme is our previous work, which can achieve a high cache hit rate in different traffic environments and cache sizes. Moreover, the PipeCache scheme is perfectly suitable for caching rules in MFTs [12]. The entries stored in the SRAM's cache table are the correspondence between flow ID and actions, which is also perfectly suitable for MFTs [10]. If the incoming packet cannot find a match in the TCAM or SRAM, it will be sent to the MFTs in the software. The cache table entries in the TCAM or SRAM will be updated by the matching results in the MFTs. Every cache miss will trigger a cache update operation.

Algorithm 1 MixedCache algorithm

Input: p: an incoming packet;
Output: the Cache_Table_Entries in TCAM and SRAM; the actions of the incoming packet;
// n: the number of MFTs, m: the number of rules in a rule set of a flow table;
// RS: Rule Set, R: Rule;
// Avail_TCAM: available TCAM entries, Avail_SRAM: available SRAM entries;
1. for each RSi in MFTs (from RS0 to RSn-1) do
2. Cover-Set (RSi);
3. while a packet p comes do //when a packet comes:
4. if p belongs to small flows:
5. for each RSi in TCAM (from RS0 to RSn-1) do // search in cache
6. for each Rj in RSi (from R0 to Rm-1) in descending priority order: do
7. if p matches with Rj then
8. Add 1 to the counter of Rj;
9. Update the LRU identifier of Rj;
10. Continue;
11. else if p cannot find a match in Rj then
12. for each RSj in software (from RS0 to RSn-1) : do // search in software
13. for each Rj in RSj (from R0 to Rm-1) in descending priority order: do
14. if p matches with Rj then
15. Set the counter of Rj to 1;
16. Update the LRU identifier of Rj;
17. Rules_to_cache = Rj + Rj.direct_children;
18. if Avail_TCAM > the number of Rules_to_cache then
19. add Rules_to_cache in TCAM;
20. else
21. delete the last recently used R for the number of Rules_to_cache times;
22. add Rules_to_cache in TCAM;
23. Continue;
24. else if p belongs to large flows:
25. if p matches with Ri in SRAM:
26. Add 1 to the counter of Ri;
27. Update the LRU identifier of Ri;
28. Continue;
29. else:
30. for each RSi in software (from RS0 to RSn-1) : do // search in software
31. for each Rj in RSi (from R0 to Rm-1) in descending priority order: do
32. if p matches with Rj then
33. Add the action to action_set;
34. Flow ID = hash (the flow that the packet comes from);
35. Creating a new entry; // the correspondence between flow ID and actions
36. if Avail_SRAM > 0;
37. Add the new entry to SRAM;
3.3. The cache manager

The cache manager is used to determine whether the incoming packet belongs to a large flow or a small flow. It sends the packets belonging to the small flows to the TCAM and sends the packets belonging to the large flows to the SRAM. The large and small flows can be judged by network measurement. The sketch is a popular method in network measurement [27-29]. In our design, we choose the most used count-min sketch [27] as the measurement method. It can query the heavy hitter at a relatively low cost. The count-min-Sketch data structure is a two-dimensional array count with width w, depth d, and d independent hash functions h1, ..., hd. The parameter K is a threshold for distinguishing between large and small streams.

When a packet comes, the cache manager performs the hash calculation on the packet's information by all hash functions. Then the cache manager adds 1 to the value of the corresponding row of each hash calculation result. Since the hash calculation has the problem of hash collision, each hash function's statistical result is relatively large. Therefore, we take the smallest value among the statistical results of all hash functions as the current flow count result. TCAM processes it before a flow is judged as a large flow. Before a packet is judged to belong to a large flow, it is processed by TCAM. When it is judged to belong to the large flow, the cache manager converts the original wildcard match rules into an exact match rule and stores it in SRAM and removes the original wildcard match rules in TCAM. Packets belonging to large flows are handed over to SRAM for processing.

4. Evaluation

In this section, we presented the simulations of our MixedCache and the strawman algorithms. We compare our proposed algorithms with the exact-match scheme and our previous work, PipeCache, a wildcard-match scheme based on MFTs.

4.1. Experiment Setup

To verify the performance of our MixedCache in the context of OpenFlow12 tuples, we still use our adapted version of ClassBench [30] and ClassBench-ng [31], which introduced in our previous work PipeCache [12]. We can use this tool to generate OpenFlow traces. The traces generated by the trace generator of ClassBench still follow the Pareto distribution [30]. In the Pareto distribution function, two parameters a and b, are used to adjust the flow distribution. To fit the flow distribution of Internet traffic, the parameter a is set to 1. The parameter b is used to adjust the locality of the generated traces. From 0 to 1, a greater value of b means higher traffic locality.

Firstly, we use ClassBench-ng to generate multiple-stage OpenFlow rules files based on two seed files with different sizes in three scales of 1 k, 5 k, and 10 k. Then, we generate ten sets of data on each scale to take the average as the results. We then set the locality parameter b from 10% to 90%, increasing by 10% each time to generate different trace files with different localities. Each trace file is about ten times the size of the corresponding ruleset. The TCAM size is also set as a parameter, which scales from 10% to 100% of the total rules number.

4.2. Experiment Results

To evaluate the performance of MixedCache, we compare it with the exact-match scheme and our previous work PipeCache based on wildcard match for MFTs in terms of the cache hit rate. Here we call the exact match scheme MicroCache for simplicity.

The threshold K for judging the large and small flows is critical to the performance of MixedCache. We test the cache hit rates of the MixedCache under different cache sizes (% rule set) with threshold K from 1 to 10. In this paper, the cache size means the ratio of the size of SRAM and the ruleset. We choose three typical images to show the test results, which is shown in figure 2. As the K increases under
the same cache size, the cache hit rate first increases and then decreases. The three devices showed the same trend. Among them, the Intel FM6000 with the highest TCAM ratio had the highest cache hit rate. Furthermore, the K with the highest cache hit rate decreases as the cache size increases.

The SRAM is used to store large flows, which are the longest first N flows in the traffic. The larger the SRAM size, the greater the number of large flows that can be stored. This means that under the same traffic environment, the requirement for a flow to be judged as a large flow becomes lower, which means that K becomes small.

In addition to cache size, we also test the impact of traffic locality on the threshold K. Figure 3 shows the test results. Under the same traffic locality, as K increases, the cache hit rate first increases and then slowly decreases. The three devices still show the same trend, and Intel FM6000 still has the highest cache hit rate. We can conclude that there is no correlation between K and the locality of the flow through the experimental results. In other words, the setting of K is only related to the configuration of hardware storage resources in the switch.
Next, we test the three schemes' performance on these three devices under the same cache size and the same traffic environment. The cache size is set to 10% of the whole ruleset. And the locality parameter $b$ is set to 0.1. In figure 4a, we can see that our MixedCache has the highest cache hit rate among different devices. Figure 4b shows the improvement ratio of MixedCache compared with other schemes on three devices. Compared with MicroCache, the improvement ratios are: 5.61%, 4.90%, 7.45%. And compared with PipeCache, the improvement ratios are: 5.15%, 5.74%, 4.94%.

The size of the cache plays an essential role in the performance of rule-caching schemes. We set the cache size from 10% to 100% of the rule set, increasing by 10% each time to test the performances of the three schemes with different cache sizes. We use the Intel FM6000 as the test device. As shown in...
Figure 5, the cache hit rates of the three schemes increased as the cache sizes increased. When the locality parameter $b$ was 0.1, the cache hit rate of MixedCache started from 56.60% with the cache size 10% and increased to 80.25% when the cache size was 100%. When the locality parameter $b$ was 0.2, the cache hit rate of MixedCache went from 61.10% to 81.90%. The other two schemes also exhibited the same trend that when the traffic had a higher locality, the caching scheme has better performance. The MixedCache outperformed the other two schemes with all cache sizes.

![Fig.5 Cache hit rates with different cache sizes.](image)

The improvement ratio of MixedCache with different cache sizes is shown in figure 6. Compared with MicroCache, the improvement ratio of MixedCache shows a downward trend. When the cache size starts from about 40%, the improvement ratio stabilizes at a relatively low level. However, compared with PipeCache, the improvement rate of MixedCache continues to increase. This is because PipeCache can only use TCAM that supports wildcard match. The size of TCAM is much smaller than that of SRAM, so the increase in hardware storage resources used for PipeCache is much smaller than that of MixedCache.

![Fig.6 Improvement ratios with different cache sizes.](image)

As shown in Figure 7, the change in traffic locality can influence the performance of the three schemes. To show the performance of different schemes under different traffic localities, we set the locality parameter from 0.1 to 0.9, increasing 0.1 each time. We repeated the test with the cache rate at 10% and 20%. The cache hit rate increases as the locality of traffic increases. Additionally, MixedCache still achieves the best performance under various traffic localities. With the increase of traffic locality, the performance of MixedCache and PipeCache are closer.
Figure 8 shows the improvement ratio of MixedCache with different traffic locality parameter values. Compared with MicroCache, the improvement ratio shows a trend of first increasing and then decreasing, and the highest improvement ratio is 15.61% and 14.10% when b=0.1 and 0.2, respectively. Compared with PipeCache, the improvement ratio of MixedCache has been slowly decreasing. This is because when the traffic locality is high enough, several wildcard rules are sufficient for matching most flows. At this time, the improvement ratio of MixedCache to PipeCache is no longer significant.

5. Conclusion
In this paper, we proposed MixedCache, an innovative and efficient rule caching scheme designed for the OpenFlow switches with MFTs. To take the advantages of wildcard match and exact match, MixedCache combines the TCAM and SRAM in the switch. To make the most use of hardware storage resources, we allocated packets by the flow characteristics. To the best of our knowledge, this is the first attempt to adopt both TCAM and SRAM on the MFTs of OpenFlow switches. Experiment results show that our design MixedCache improves cache hit rate by up to 15.61% compared to the exact-match scheme and by up to 29.69% compared to our previous work PipeCache based on the wildcard-match scheme.

Acknowledgments
This work was supported in part by grants from the National Key R&D Program of China (No. 2018YFB1800402), the Open Fund of PDL (No. WDZC20205500110), and the National Natural Science Foundation of China (Grant No.61702538).
References

[1] Esch J. Prolog to, "software-defined networking: a comprehensive survey"[J]. Proceedings of the IEEE, 2015, 103(1):10-13.

[2] McKeown N, Anderson T, Balakrishnan H, et al. OpenFlow: Enabling Innovation in Campus Networks[J]. ACM Sigcomm Computer Communication Review, 2008, 38(2):69-74.

[3] Li C, Li T, Li J, et al. Memory Optimization for Bit-Vector-Based Packet Classification on FPGA[J]. Electronics, 2019, 8(10).

[4] Shi Z, Yang H, Li J, et al. MsBV: A Memory Compression Scheme for Bit-Vector-Based Classification Lookup Tables[J]. IEEE Access, 2020: 38673-38681.

[5] Li C, Li T, Li J, et al. Enabling Packet Classification with Low Update Latency for SDN Switch on FPGA[J]. Sustainability, 2020, 12(8): 3068.

[6] Song H, Dharmapurikar S, Turner J, et al. Fast hash table lookup using extended bloom filter: an aid to network processing[J]. ACM SIGCOMM Computer Communication Review, 2005, 35(4): 181-192.

[7] Salisbury B. TCAMs and OpenFlow-what every SDN practitioner must know[J]. See http://tinyurl.com/kjy99uw, 2012.

[8] Kogan K, Nikolenko S, Rottenstreich O, et al. SAX-PAC (scalable and expressive packet classification)[C]//Proceedings of the 2014 ACM conference on SIGCOMM. 2014: 15-26.

[9] Cheng T, Wang K, Wang L C, et al. An in-switch rule caching and replacement algorithm in software defined networks[C]//2018 IEEE International Conference on Communications (ICC). IEEE, 2018: 1-6.

[10] Pfaff B, Pettit J, Koponen T, et al. The design and implementation of open vswitch[C]//12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15). 2015: 117-130.

[11] Shelly N, Jackson E J, Koponen T, et al. Flow caching for high entropy packet fields[C]//Proceedings of the third workshop on Hot topics in software defined networking. 2014: 151-156.

[12] Yang J, Li T, Yan J, et al. PipeCache: High Hit Rate Rule-Caching Scheme Based on Multi-Stage Cache Tables[J]. Electronics, 2020, 9(6): 999.

[13] Fundation O N. The benefits of multiple flow tables and ttps[R]. ONF Technical Report, 2015.

[14] Firestone D. {VFP}: A Virtual Switch Platform for Host {SDN} in the Public Cloud[C]//14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17). 2017: 315-328.

[15] Firestone D. {VFP}: A Virtual Switch Platform for Host {SDN} in the Public Cloud[C]//14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17). 2017: 315-328.

[16] Yan B, Xu Y, Xing H, et al. Cab: A reactive wildcard rule caching system for software-defined networks[C]//Proceedings of the third workshop on Hot topics in software defined networking. 2014: 163-168.

[17] Katta N, Alipourfard O, Rexford J, et al. Cacheflow: Dependency-aware rule-caching for software-defined networks[C]//Proceedings of the Symposium on SDN Research. 2016: 1-12.

[18] Sheu J P, Chuo Y C. Wildcard rules caching and cache replacement algorithms in software-defined networking[J]. IEEE Transactions on Network and Service Management, 2016, 13(1): 19-29.

[19] Li X, Xie W. CRAFT: A cache reduction architecture for flow tables in software-defined networks[C]//2017 IEEE Symposium on Computers and Communications (ISCC). IEEE, 2017: 967-972.

[20] Li R, Pang Y, Zhao J, et al. A Tale of Two (Flow) Tables: Demystifying Rule Caching in OpenFlow Switches[C]//Proceedings of the 48th International Conference on Parallel Processing. 2019: 1-10.

[21] Grigoryan G, Liu Y. Pfca: a programmable fib caching architecture[C]//Proceedings of the 2018
Symposium on Architectures for Networking and Communications Systems. 2018: 97-103.

[22] Broadcom BCM56846 StrataXGS 10/40 GbE Switch. http://www.broadcom.com/products/features/BCM56846.php.

[23] Stephens B, Cox A, Felter W, et al. PAST: Scalable Ethernet for data centers[C]//Proceedings of the 8th international conference on Emerging networking experiments and technologies. 2012: 49-60.

[24] Naous J, Erickson D, Covington G A, et al. Implementing an OpenFlow switch on the NetFPGA platform[C]//Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems. 2008: 1-9.

[25] Naous J, Gibb G, Bolouki S, et al. NetFPGA: reusable router architecture for experimental research[C]//Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow. 2008: 1-7.

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

[27] Cormode G, Muthukrishnan S. An improved data stream summary: the count-min sketch and its applications[J]. Journal of Algorithms, 2005, 55(1): 58-75.

[28] Yang T, Jiang J, Liu P, et al. Elastic sketch: Adaptive and fast network-wide measurements[C]//Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. 2018: 561-575.

[29] Tang L, Huang Q, Lee P P C. MV-Sketch: A fast and compact invertible sketch for heavy flow detection in network data streams[C]//IEEE INFOCOM 2019-IEEE Conference on Computer Communications. IEEE, 2019: 2026-2034.

[30] Taylor D E, Turner J S. Classbench: A packet classification benchmark[J]. IEEE/ACM transactions on networking, 2007, 15(3): 499-511.

[31] Matoušek J, Antichi G, Lučanský A, et al. Classbench-ng: Recasting classbench after a decade of network evolution[C]//2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS). IEEE, 2017: 204-216.