FDRC: Flow-Driven Rule Caching Optimization in Software Defined Networking

He Li*, Song Guo*, Chentao Wu†, and Jie Li†,

*School of Computer Science and Engineering, The University of Aizu
Aizuwakamatsu, Fukushima 965-8580 Japan
†Shanghai Key Laboratory of Scalable Computing and Systems, Department of Computer Science & Engineering
Shanghai Jiao Tong University, Shanghai, China 200240
{d8141105, sguo}@u-aizu.ac.jp, {wuct, lijie}@cs.sjtu.edu.cn

***Abstract*—With the sharp growth of cloud services and their possible combinations, the scale of data center network traffic has an inevitable explosive increasing in recent years. *Software defined network* (SDN) provides a scalable and flexible structure to simplify network traffic management. It has been shown that *Ternary Content Addressable Memory* (TCAM) management plays an important role on the performance of SDN. However, previous literatures, in point of view on rule placement strategies, are still insufficient to provide high scalability for processing large flow sets with a limited TCAM size. So caching is a brand new method for TCAM management which can provide better performance than rule placement. In this paper, we propose FDRC, an efficient flow-driven rule caching algorithm to optimize the cache replacement in SDN-based networks. Different from the previous packet-driven caching algorithm, FDRC is characterized by trying to deal with the challenges of limited cache size constraint and unpredictable flows. In particular, we design a caching algorithm with low-complexity to achieve high cache hit ratio by prefetching and special replacement strategy for predictable and unpredictable flows, respectively. By conducting extensive simulations, we demonstrate that our proposed caching algorithm significantly outperforms FIFO and *least recently used* (LRU) algorithms under various network settings.**

I. Introduction

As one of the most significant technologies for large-scale data center network, *Software Defined Network (SDN)* demonstrates great potentiality on management, scalability and other features. SDN-enabled switches, managed by a logically centralized controller, support fine-grained and flow-level controls of data center networks. Such controls are desirable with flexible policies under programmable configuration and visible flow management [1]–[4]. Typically, the flow-based control is implemented by installing simple packet-processing rules in the underlying switches. These rules can match the head of packet-header fields in the network flows, and perform some actions, such as forwarding, modifying, or sending to controller for further processing. For each flow, combining multiple policies can provide a flexible, fine-grained and dynamical control. However, with the increasing number of flows in data center networks, the flow-based control leads to a combinatorial explosion in terms of all number of rules per switch.

In previous SDN switches, *Ternary Content Addressable Memory* (TCAM) is a typical memory to store rules, which can compare an incoming packet to the patterns in all of rules at a line rate simultaneously [5]. However, TCAM is not a cost-effective way to provide high performance. First, compared to the ordinary RAM, TCAM needs approximate 400 times monetary cost and consumes 100 times power. Second, even in high-end commodity switches, due to the limited size of TCAM, the space cannot contain a large number of various rules. Furthermore, the updating speed of rules is slow in TCAM, which supplies around 50 rule-table updates per second. This is a major restriction for adopting policies to support flow-based control in large-scale networks [6].

Rule placement optimization is an existing method to improve the processing capacity of data center networks [7]–[9]. The mechanism of this method is that by getting the information on the whole network, after some analysis on all existing flows, a proper placement strategy can be found to improve the flow processing capacity. However, these optimizing strategies are usually static with a limited number of flows. Furthermore, when the status of flows changes, such as flow destination movement, traffic variation, etc., updating the rules in all switches is unaffordable.

Unlike rule placement optimization which regards rule space as a limited resource, rule caching strategies efficiently use space to store the rules and cache the most frequent rules in TCAM as caching [10]–[12]. Therefore, all rules can be handled in the network via replacement policies and the performance can be enhanced in terms of high hit ratio. Compared to the rule placement optimization methods, rule caching is a better approach to enable the flow-based control to provide both high performance and scalability, especially in a large-scale data center network.

Ordinary caching algorithms, such as *Least Recently Used* (LRU), are not appropriate to the flow-based control in large-scale data center networks. One reason is that these algorithms are based on single rule replacement for each switch, while proper control policies should be based on a global view of multiple rules in all switches. Furthermore, the fact that flow traffic is predictable and correlated with time should be exploited. Through acquiring and analyzing the history flow information, an SDN controller can easily get the significant information of applications, i.e., the prediction of flow traffic

978-1-4673-6432-4/15/$31.00 ©2015 IEEE

distribution, such that higher performance can be archived.

To address this problem on rule caching, in this paper, we model the optimization problem and design a caching algorithm based on the prefetching and replacement strategies. This algorithm achieves high hit ratio by replacing rules with the flow forwarding paths and replacing them integrally with different replacement strategies for predictable and unpredictable flows.

The main contributions of this paper are summarized as follows,

- First, we study a caching optimization problem for flow-based control in the data center networks. This problem is challenging because of the flow variability and the constrained cache space in SDN switches.
- Second, we propose an efficient heuristic algorithm to solve the caching optimization problem. Our basic idea is to design two different replacement strategies for predictable and unpredictable flows.
- Finally, extensive simulations are conducted and the results show that the proposed algorithm can significantly increase the hit ratio and thus improve the network performance.

The rest of this paper are organized as follows. Our network model and system design are introduced in Section II. Section III presents our algorithm design. Section IV gives simulation results. Finally, Section V concludes this paper.

II. System Model

In this section, we first discuss the rule caching and flow-based control in SDN-based network. Then, we state the main problem in the rule caching. For better understanding, we use Table I to show the meanings of major notations.

Table I: Notations

Notation	Description
f_i	Flow i
F	Flows in the network
f_i	Flow i
R	Rules in the network
r_i	Rule i of flow i
S	Switches in the network
S_i	Switches in the forwarding path of flow i
s_j	Switch j
B_j	Cache size of switch j
$f_i(t)$	Network traffic density function of flow i at time t
X_{ij}	Whether switch j caches rule of flow i
$h_i(t)$	Cache Hits of rule i at time t
$H_i(t)$	Cache hits of rule i from time 0 to t
$F_i(t)$	Traffic of flow i from time 0 to t
$C_i(t)$	Cache hit ratio of flow i from time 0 to t
$C(t)$	Cache hit ratio of entire network
$T(f_i, t)$	Time to the next coming packet of f_i at time t
$T_{\max}$	Maximum waiting time for the next packet coming

A. Flows and Rule Caching

Unlike traditional networks, SDN considers network flows as the basic units and control methodologies are based on the flows in typical. However, the rule updating in SDN switches is related to the network packets. A rule updating takes place during the processing of new network packets. When a new packet is checked and no matching field with the entries of flow tables in the switch, the packet will be sent to the controller for further processing. In general, after the processing of a new packet, the related rules are updated in the switch. This strategy is considered as a FIFO replacement. FIFO is not a good replacement algorithm because that some rules for processing rare flows can stay in the TCAM for a long period. In a general network, many flows only have several packets in a short period of time. By using FIFO replacement policy, the rules of these flows cost too much TCAM space with a low cache hit ratio.

Least Recently Used (LRU) is an advanced caching algorithm used in many fields. Exiting solutions also use LRU for SDN caching replacement. However, it is not an appropriate caching algorithm in SDN since LRU is still a packet-driven algorithm. To illustrate the LRU in flow-based control SDN, we use an example shown in Figure 1, where three flows, f_1, f_2 and f_3 are processed using rules r_1, r_2 and r_3, respectively. Suppose each switch cache can store two rules. For simplifying the problem, we consider the traffic the traffic of three flows regularly distributes in the time-domain cannot be interrupted (e.g., during the period from t_2 to t_4). Initially, since there is no rule cached in the TCAM space, there are two cache misses when flow f_1 and f_3 come. After that, when $f2$ comes to the switch , the algorithm replaces r_3 to r_2 with a cache miss because r_3 is the least recently used rule. However, since the LRU algorithm does not know that f_3 will come back soon, r_3 needs to be re-deployed to replace r_1. Finally, from the result of LRU replacement, the cache hit ratio in this example is 0.

Another problem is that most of flows in a data center network are relevant with more than one switches. While existing caching algorithm is packet-driven, the replacement only occurs in a single switch. When some other switches forward a same packet, controller need to process this packet again. In an SDN structure, all switches are managed by a centralized controller and the controller can also get the traffic information of each flow in the network. As a result, a flow-driven caching is more appropriate than a packet-driven algorithm with SDN. That's why we propose a novel rule caching algorithm to state the flow-driven caching problem.

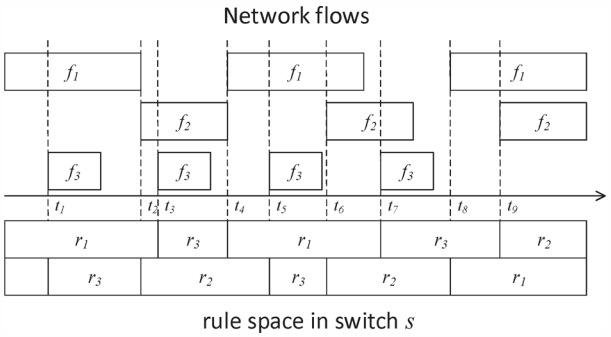


Figure 1: Caching rules for network flows

B. Rule Caching Problem

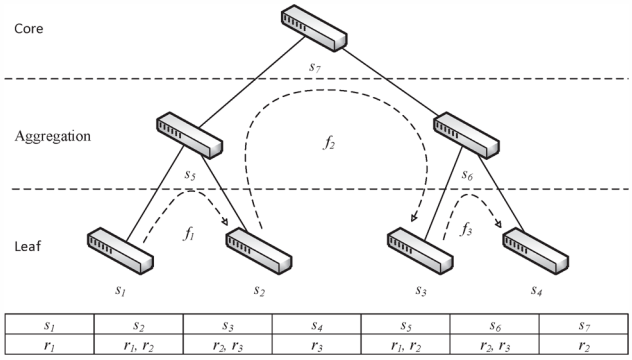


Figure 2: The network flows and rule caching in a typical data center topology

As shown as Figure 2, we consider a data center network consisting of a set $S = \{s_1, s_2, ..., s_m\}$ of switches which includes leaf switches, aggregation switches and core switches. For any switch s_j in S, it maintains a TCAM-based flow table which can cache at most B_j forwarding rules.

We consider set $F = \{f_1, f_2, ..., f_n\}$ of network flows, with an associated set $R = \{r_1, r_2, ..., r_n\}$ of forwarding rules among SDN switches. Let S_i denote the set of switches in the forwarding path of f_i, i.e, any switch in S_i maintains r_i. For example, $S_1 = \{s_1, s_2, s_5\}$.

In this paper, we investigate a rule caching problem under our system model by addressing the following two challenges. First, rule capacity of each switch is limited, and the rules required by a network flow may be forwarded by multiple switches, or even cannot be cached by the SDN-based network. We define a variable X_{ij} to denote the rule caching as follows.

$$X_{ij} = \begin{cases} 1 & r_i \text{ is cached in } s_j \\ 0 & r_i \text{ is not cached in } s_j \end{cases} \quad (1)$$

The cache capacity constraint at each switch can be represented by:

$$\sum_{i=1}^{n} X_{ij} \leq B_j. \quad (2)$$

Letting $f_i(t)$ be the traffic density of flow f_i, we can define the cache hits at time instance as follows.

$$h_i(t) = f_i(t) \sum_{s_j \in S_i} X_{ij} \quad (3)$$

Therefore, the cache hits $H_i(t)$ from time 0 to t can be expressed as:

$$H_i(t) = \int_0^t h_i(t)dt = \int_0^t f_i(t) \sum_{s_j \in S_i} X_{ij} dt. \quad (4)$$

Finally using $F_i(t)$ to denote the overall traffic from time 0 to t, i.e.,

Algorithm 1 Flow-driven caching algorithm

Require: In time t, a packet of flow f_i comes to the switch.
1: **for** $s_j \in S_i$ **do** ▷ Traverse switches in the forwarding path
2: **if** r_i is not (i.e., $X_{ij} = 0$) cached in s_j **then**
3: **if** $\sum_{i=1}^{n} X_{ij} = B_j$ **then** ▷ Cache replacement
4: $m = \arg\max_{k \quad X_{kj}=1} T_k$;
5: remove r_m from cache;
6: **end if**
7: put r_i in cache;
8: Set timer T_i using Algorithm 2
9: **end if**
10: **end for**

$$F_i(t) = \int_0^t f_i(t)dt, \quad (5)$$

we can get the hit ratio $C_i(t)$ from 0 to t as follows.

$$C_i(t) = \frac{H_i(t)}{F_i(t)} = \frac{\int_0^t f_i(t) \sum_{s_j \in S_i} X_{ij} dt}{\int_0^t f_i(t)dt} \quad (6)$$

Similarly, we define $C(t)$ to denote the total hit ratio shown in (7).

$$C(t) = \frac{\sum_{i=1}^{n} H_i(t)}{\sum_{i=1}^{n} F_i(t)} = \frac{\sum_{i=1}^{n} \int_0^t f_i(t) \sum_{s_j \in S_i} X_{ij} dt}{\sum_{i=1}^{n} \int_0^t f_i(t)dt} \quad (7)$$

The problem of rule caching in SDN-based network: given a software defined network, a set of network flows with rules and a period, the rule caching problem attempts to find a part of flows and put rules of these flows to the cache of each SDN switch of the network to maximum the accumulated number of cache hits in this period.

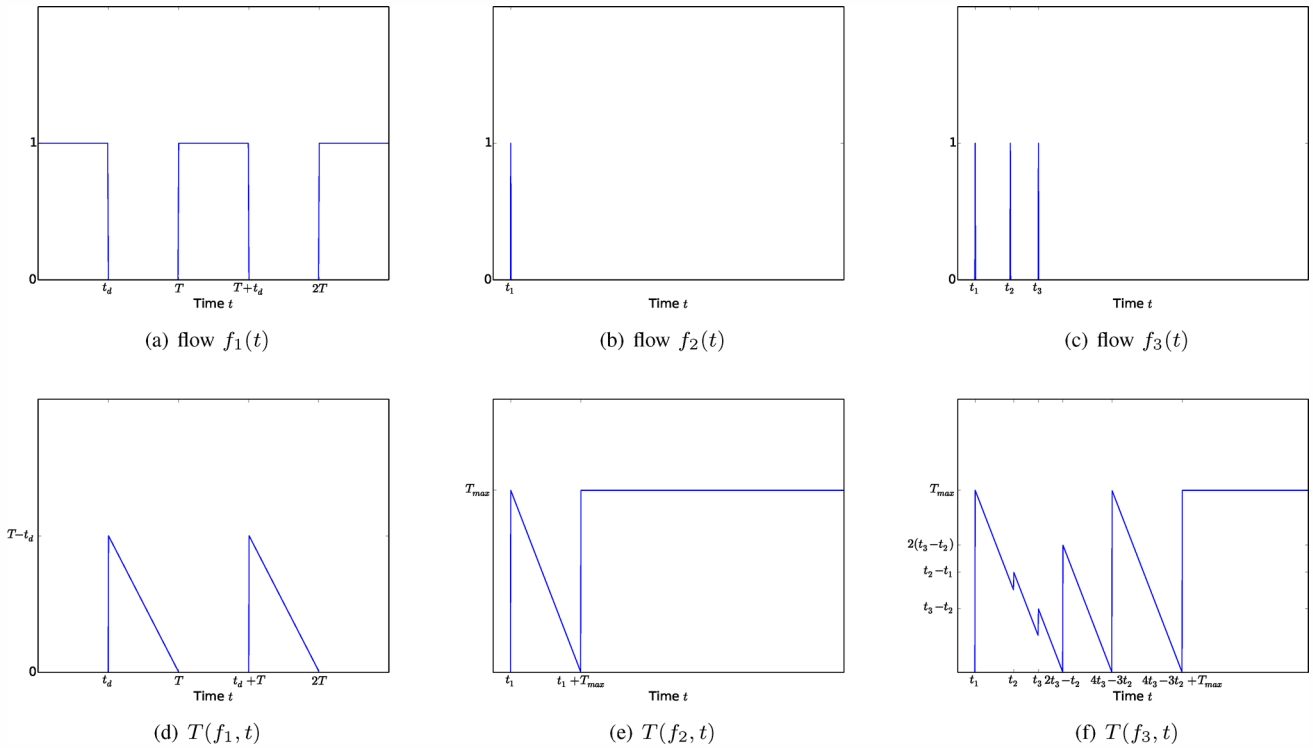
III. Algorithm

In this section, we propose a *Flow-Driven Caching* (FDRC) algorithm to solve the rule caching problem by taking into account of both traffic pattern and routing path of each flow. The basic idea of our algorithm is as follows.

1) When the rule for any flow f_i needs to be cached on a switch, it should be cached over all switches over the entire path of the flow, i.e., rule prefetching.
2) A timer T_i is associated to the entry of flow f_i with a value of the estimated time to the next hit of the entry. When an entry replacement happens at a switch, the entry with maximum timer value in that switch will be chosen.

The description of FDRC is given in Algorithm 1.

When setting timer T_i, we consider two types of flow patterns: predictable flows (e.g., the flows from deterministic network service) and unpredictable flows (e.g., spontaneous traffic). For the former case, the timer is simply set as the



(a) flow $f_1(t)$ (b) flow $f_2(t)$ (c) flow $f_3(t)$

(d) $T(f_1,t)$ (e) $T(f_2,t)$ (f) $T(f_3,t)$

Figure 3: Examples of $T(f_i,t)$ for various types of flows

Algorithm 2 Setting timer T_i for unpredicted flow f_i

```
Start timer T_i with an initialized value T_max;
while 1 do
    if a new packet of flow f_i comes before T_i expires then
        set ΔT as the latest packet arriving interval;
        start T_i with value ΔT;
    end if
    if T_i expires then
        if ΔT = T_max then
            freeze T_i with value T_max;
            break;
        end if
        start T_i with value min(2ΔT, T_max);
    end if
end while
```

time to next packet arrival. For the latter case, the estimation of the time is updated using Algorithm 2.

From Algorithm 2, we notice that when the estimated next arrival is earlier than expected, the timer should be updated by the latest arriving interval; otherwise, the timer should be reset with a doubled value, but no exceeding $T_{\max}$, after expiration.

To better understanding how T_i behaves, we show its value as a function of time t, denoted as $T(f_i,t)$, using examples in Figure 3. In Figure 3(a)-(c), values 1/0 on y-axis indicate if a flow is active or not at any time t. The corresponding functions $T(f_i,t)$ are illustrated in Figure 3(d)-(f), respectively.

Figure 3(b) shows a predictable flow f_1, which keeps active for a period t_d and then silent for $T-t_d$, periodically. Therefore, $T(f_1,t)$ in Figure 3(d) is also a periodic function, with a cycle length T, that can be determined as follows.

$$T(f_1,t)=\begin{cases}0 \ t\in(nT,nT+t_d)\\(n+1)T+t_d-t \ t\in(nT+t_d,(n+1)T)\end{cases}$$
$$n=0,1,2,...$$

The unpredictable flows are given in Fig. 3(b) and (c). For example, when a packet of f_2 arrives at t_1 shown in Figure 3(b), timer T_2 is initialized as $T_{\max}$, i.e.,

$$T(f_2,t)=T_{\max}+t_1-t.$$

According to Algorithm 2, after T_2 expires at $T_{\max}+t_1$, $T(f_2,t)$ remains $T_{\max}$ as shown in Figure 3(e). On the other hand, when a packet of f_3 arrives at t_2 earlier than the estimated time, $T(f_3,t)$ should be updated from $T(f_3,t)=T_{\max}+t_1-t$ to $T(f_3,t)=(t_2-t_1)+t_2-t$ as shown in Figure 3(f). Later on, T_3 will expire at $2t_3-t_2$ and restart from $\min(2(t_3-t_2),T_{\max})=2(t_3-t_2)$. The second expiration happens at $4t_3-3t_2$ and T_3 restarts from $\min(4(t_3-t_2)),T_{\max})=T_{\max}$. Eventually, it remains at $T_{\max}$.

IV. Evaluation

We conduct simulation based experiments to evaluate the performance of the proposed algorithm. Simulation results

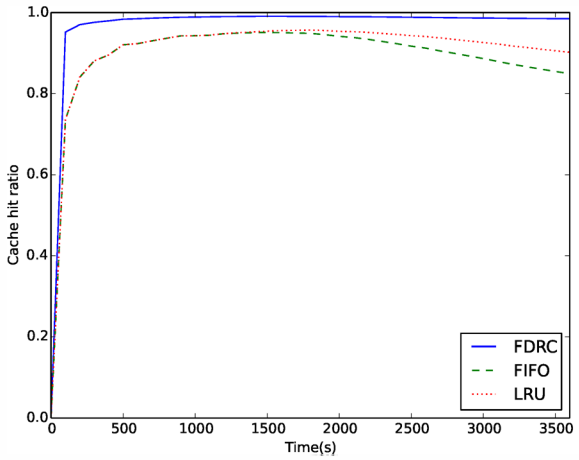


Figure 4: Cache hit ratio in the first hour

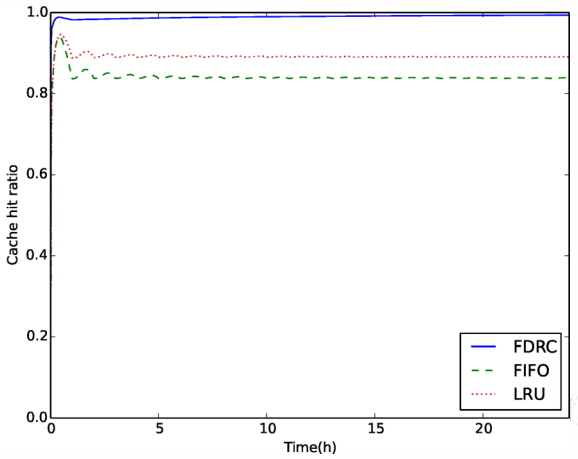


Figure 5: Cache hit ratio per one day period

under different network parameters are presented.

A. Simulation Settings

To evaluate the performance, we compare the cache-hit of our algorithm to other popular algorithms FIFO and LRU. The simulation is conducted over a number of randomly generated networks by using a python 2.7 script with the network library 1.6 on a desktop computer. We use two types of flows in the simulation, flows with periodic traffic and random traffic. For the periodic traffic, the period T of the traffic cycles is uniformly distributed within the range $[2s, 100s]$ and traffic duration time in each cycle is also uniformly distributed within range $[1s, T]$. For the random traffic, the packets are randomly generated with uniformly distributed arrival interval over the whole simulation time. Each flow is associated with a group of switches for a normally distributed size in range $[1, 10]$. In the default simulation settings, the cache size of each SDN-enabled switch is uniformly distributed in range $[15, 25]$, and the ratio of predictable flows in total flows is 40%. All simulation results are averaged over 20 network instances.

B. Simulation Results

We first evaluate the cache-hit-ratio of all algorithms under the default settings. As shown in Figure 4 , when new flows are injected into the network in the first hour of simulation, caching rules can significantly improve the hit ratio rapidly for all algorithms. Moreover, our FDRC algorithm always has better performance than the other two algorithms, in terms of both maximum ratio and how fast this ratio can be archived. After 1500s, FIFO has the lowest hit ratio in these three algorithms, and LRU can outperform FIFO by up to 6%. We then extend our simulation in a longer period as shown in Figure 5. After the cache of each switch becomes full, the hit ratio of both FIFO and LRU decreases due to their improper replacement strategies. Eventually, the ratio of FDRC, FIFO and LRU become stable at 99.4%, 83.8% and 89.0%, respectively.

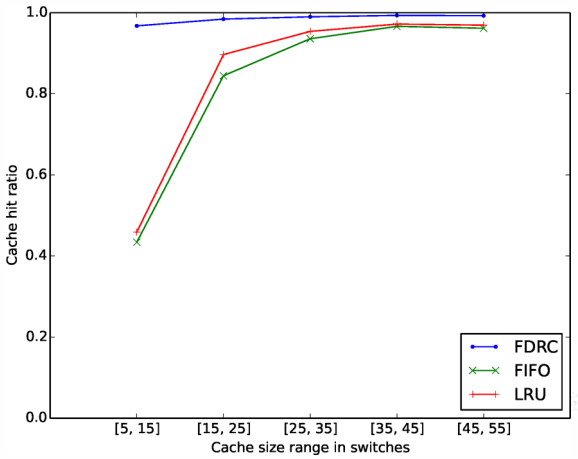


Figure 6: Cache hit ratio with different cache size in switches

We then investigate how the cache size effects the cache-hit-ratio using five ranges of cache size: $[5, 15]$, $[15, 25]$, $[25, 35]$, $[35, 45]$ and $[45, 55]$. As shown in Figure 6, the FDRC algorithm achieves much higher cache-hit-ratio than the other two algorithms in the cases with small cache size. When the cache size is in the range $[5, 15]$, the FDRC algorithm achieves a cache-hit-ratio over 90%, while the ratios of the other two are both below 50%. With an increased cache size, the gap among these algorithms becomes smaller. for example, under the range $[45, 55]$, FDRC improves the performance of the other two algorithms by 3.2% only. This is because little cache replacement happens when sufficient cache is available. In summary, the cache size have a critical impact to the hit ratio of the traditional cache algorithms. In contrast, FDRC is less sensitive to the cache size since it adopts the prefetching strategy.

Finally, we study the impact of various types of flows to the performance by adjusting the ratio of predictable flows in

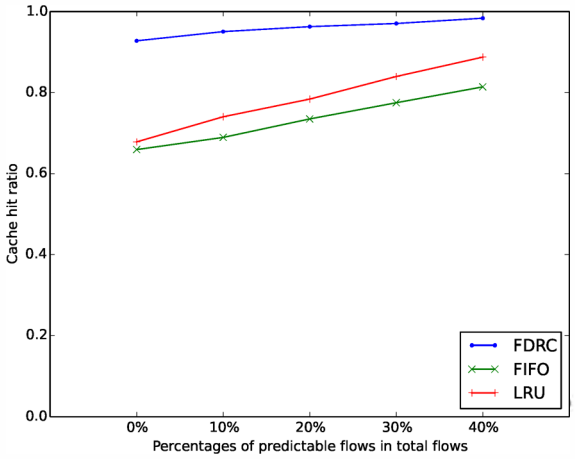


Figure 7: Cache hit ratio with different percentages of predictable flows in all flows

total flows. When this ratio is 0, only unpredictable flows are generated in our simulation. From the results shown in Figure 7, we observe that FDRC always achieves the highest cache-hit-ratio, especially in the case that the unpredictable flows dominate. For example, when only unpredictable flows exist, FDRC achieves a ratio over 90%, while both FIFO and LRU below 50%. With more predictable flows in the network, the hit ratio increases for all three algorithms. When the percentage of predictable becomes 40%, the cache hit ratio of FDRC is still 10% and 15% higher than that of LRU and FIFO, respectively.

V. Conclusions

In this paper, we propose a rule caching model based on the traffic and path of flows to optimize replacement of a switch cache. We apply the rule prefetching over the entire path of each flow to reduce the cache miss. We also design special processing on both predictable and unpredictable flows. Finally, extensive simulations are conducted to show that the proposed caching algorithm can significantly increase the hit ratio than traditional algorithms.

Conclusions

This work is partially sponsored by the National Basic Research 973 Program of China (No. 2015CB352403), the National Natural Science Foundation of China (NSFC) (No. 61261160502, No. 61272099, No. 61303012, No.61332001), the Program for Changjiang Scholars and Innovative Research Team in University (IRT1158, PCSIRT), the Scientific Innovation Act of STCSM (No. 13511504200), the Shanghai Natural Science Foundation (No. 13ZR1421900), the Scientific Research Foundation for the Returned Overseas Chinese Scholars, and the EU FP7 CLIMBER project (No. PIRSESGA-2012-318939).

References

[1] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: a distributed control platform for large-scale production networks," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6.

[2] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 254–265.

[3] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: State distribution trade-offs in software defined networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 1–6.

[4] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing software-defined networks," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, ser. nsdi'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 1–14.

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[6] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using sdn," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 27–38.

[7] A. X. Liu, C. R. Meiners, and E. Torng, "Tcam razor: A systematic approach towards minimizing packet classifiers in tcams," *IEEE/ACM Trans. Netw.*, vol. 18, no. 2, pp. 490–500, Apr. 2010.

[8] C. Meiners, A. Liu, and E. Torng, "Topological transformation approaches to tcam-based packet classification," *Networking, IEEE/ACM Transactions on*, vol. 19, no. 1, pp. 237–250, Feb 2011.

[9] ——, "Bit weaving: A non-prefix approach to compressing packet classifiers in tcams," *Networking, IEEE/ACM Transactions on*, vol. 20, no. 2, pp. 488–500, April 2012.

[10] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks." in *INFOCOM*. IEEE, 2013, pp. 545–549.

[11] M. Moshref, M. Yu, A. Sharma, and R. Govindan, "Scalable rule management for data centers," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, ser. nsdi'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 157–170.

[12] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the "one big switch" abstraction in software-defined networks," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '13. New York, NY, USA: ACM, 2013, pp. 13–24.