Traffic-Aware Rule-Cache Assignment in SDN: Security Implications

Sudip Misra*, Niloy Saha*, Rupayan Bhakta†

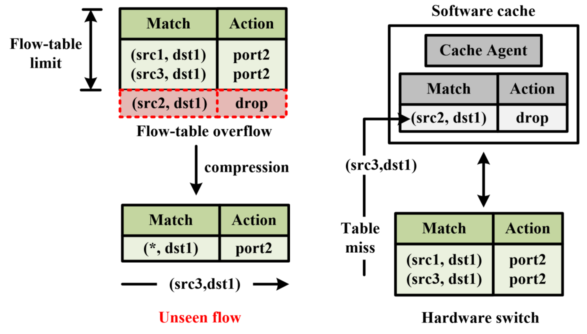
*Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, 721302, India
Email:{sudipm, niloysaha}@iitkgp.ac.in
†Department of Computer Science and Engineering, National Institute of Technology Durgapur, 713209, India
Email: rupayanbhakta2014@gmail.com

Abstract*—Cache-based flow-rule management in SDN aims to address the issue of limited ternary content addressable memory (TCAM) in switches by combining software switches (cache) with TCAM hardware. This preserves the fast packet-processing capabilities of hardware, while providing a large flow-space for rules pertaining to QoS management and security. However, to improve the reliability of such software caches, and meet requirements such as scalability and fault-tolerance, they should be placed in a distributed manner in the network. The dynamic assignment of these software caches with the hardware switches is challenging; static or improper assignment may lead to increased load imbalance, which affects overall network performance, and may make the network more vulnerable to attacks such as denial-of-service. Therefore, in this paper, we consider the traffic-aware rule cache assignment problem to lower the overall delay and network overhead. We model the problem using a many-to-many matching framework and show that the proposed algorithm arrives at a pairwise stable outcome. Extensive simulation results show that the proposed scheme reduces the average delay by 10% and 35%, and the total network overhead by 19% and 39% compared to minimum-distance based and random based cache assignment schemes, respectively.

***Index Terms*—Software-Defined Networking, Cache, Rule Management, Two-sided Matching**

I. Introduction

Software Defined Networking (SDN) enables improved control of networks by introducing networking abstractions for orchestrating data-plane elements using a centralized controller. The forwarding abstractions provided by the OpenFlow protocol [1] translate high-level policies into simple network primitives (flow-rules) in order to simplify network management. However, the match-action model followed by OpenFlow necessitates the use of TCAM hardware switches for fast processing, which limits the usefulness of SDN due to considerations of cost and power [2], [3]. In existing literature, there are two main approaches to address this issue — a) *compressing* flow-rules to reduce TCAM utilization [4], [5], and b) *caching*, where TCAM hardware switches are augmented using slower, but less expensive software switches, while faster TCAM is used as cache [6], [7]. The compression-based approaches, however, change the forwarding semantics, and can lead to security vulnerabilities, as shown in Figure 1a. The wildcard rule (*, dst1) used for compression allows the malicious flow (src2, dst1) to pass through undetected. On the other hand, caching-based approaches preserve the original policies, at the cost of additional software switches. Figure 1b shows the software cache architecture, where the TCAM hardware is augmented using software switches and a cache agent responsible for distributing flow-rules between the slow software cache and the fast TCAM.



(a) Compression-based strategies (wildcard matching) leads to unseen flows causing security risk

(b) Software cache presents the illusion of large flow-space while preserving the advantages of fast TCAM

Fig. 1: Motivating example showing security benefits of a cache-based approach

In order to improve reliability, these software caches may be placed in a distributed manner in the network, where more than one hardware switch may be connected to a software cache (scalability), and one hardware switch may be connected to more than one software cache (fault-tolerance). In a practical scenario, it is preferable to minimize the number of such software caches, in order to reduce overall costs. This, along with the many-to-many mapping constraints introduced due to scalability and fault-tolerance requirements, make the assignment of hardware switches to appropriate cache instance challenging. Moreover, in many application domains of SDN, such as cellular and IoT networks, *in-band* communication is used, where control messages are routed alongside the data-plane packets. Thus, latencies between hardware switches and cache instances may be non-uniform, which further complicates the assignment problem.

To address these concerns, we model traffic-aware rule cache assignment (TRC) as an optimization problem to reduce the overall delay and network overhead. A crucial challenge is to design an efficient algorithm to solve the TRC problem, so that the software cache instances can be dynamically re-assigned depending on varying network conditions. For this, we model the TRC problem using a *many-to-many* matching framework, where the hardware switches and cache instances are regarded as two sets of players. The key contributions of this paper are as follows:

- We formulate an integer program to determine the minimum number of software switches (caches) required to meet scalability and fault-tolerance requirements.
- We model the traffic-aware rule cache assignment (TRC) problem in SDN using a *many-to-many* matching framework to reduce the overall delay and network overhead.
- We confirm that the proposed algorithm converges to a stable outcome within a few iterations.
- Extensive simulations show that the proposed scheme significantly reduces the overall delay and network overhead.

II. Related Work

In this Section, we analyze the existing literature from two perspectives — a) cache-based flow-rule management in SDN, and b) matching-based approaches for network resource allocation.

Cache-based flow-rule management. Flow-rule caching is a promising approach for flow-rule management in SDN. In [8], the authors consider the problem of limited memory of SDN switches by using hardware TCAM as a cache for the most frequently used flow rules. The authors focused on efficient cache replacement policies in order to minimize the table-miss ratio, and decrease network overhead. This is different from the proposed scheme, where we consider distributed software caches. Katta *et al.* [6] considered a caching-based approach for SDN, and devised a method to handle wildcard rule-dependencies between the fast hardware TCAM and the slower software switches. Ruia *et al.* [7] proposed a similar cache-based solution for software-defined edge/access networks, termed *flowcache*. Flowcache acts as a transparent layer between the SDN controller and the switches, and temporarily stores the content of the recent flow-table entries. Thus, it can reduce the access time if similar flow requests arrive in the future. Both [6] and [7] consider a distributed cache architecture, but do not discuss proper assignment of the cache instances to the hardware switches. Huang *et al.* [9] consider the rule-caching problem with an aim to achieve optimal trade-off between the TCAM occupation cost and the controller overhead due to table-miss. In their work, the authors considered a hardware TCAM with single controller, and focused on configuring the appropriate timeout values for the flow-rules. In contrast, in the proposed work, we consider hardware TCAM augmented with distributed software-caches, and focus on optimal mapping between them.

Matching-based approaches for network resource allocation. Matching theory is widely used to solve resource allocation problems in networks. Han *et al.* [10] presents a survey of matching theory applications in wireless networks. Hamidouche *et al.* [11] used matching theory to address distributed edge caching in small cell networks. Wang *et al.* [12] considered a matching-theory based solution to address dynamic controller assignment in software-defined networks. The authors modeled the switch controller assignment problem as a *many-to-one* matching game. In these works, matching-based frameworks were used to develop efficient solutions to different network resource allocation problems. Inspired by this, in this work, we follow a similar approach and model the rule-cache assignment problem using a *many-to-many* matching framework.

III. System Model

In this Section, we present the system architecture of a software-cache enabled software-defined network consisting of hardware TCAM and distributed software caches, and present an integer programming formulation for the traffic-aware rule cache assignment (TRC) problem. The key notations are summarized in Table I.

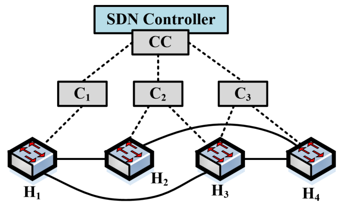


Fig. 2: Architecture

TABLE I: Summary of key notations

Notation	Description
$\mathcal{H}$	Set of hardware switches.
$\mathcal{C}$	Set of software cache instances.
q_j^s	Scalability quota of each cache instance $j \in \mathcal{C}$.
q_i^{ft}	Fault-tolerance quota of each hardware switch $i \in \mathcal{H}$.
$x(t)$	Assignment of hardware switches to cache instances in timeslot t.
$\delta(t)$	Overall delay associated with assignment $x(t)$
$o(t)$	Control overhead associated with assignment $x(t)$

Figure 2 shows the system architecture comprising of a set of hardware switches $\mathcal{H} = \{H_i \mid i \in \mathbb{Z}^+\}$, a set of software cache instances $\mathcal{C} = \{C_i \mid i \in \mathbb{Z}^+\}$, and a cache controller (CC), located at the SDN controller. The CC controls the assignment between the hardware TCAMs and the cache instances. Let $x(t)_{ij}$ denote whether hardware switch $i \in \mathcal{H}$ is assigned to cache instance $j \in \mathcal{C}$ in timeslot t ($x(t)_{ij} = 1$) or not ($x(t)_{ij} = 0$). Let q_j^s be the maximum number of hardware switches each cache instance $j \in \mathcal{C}$ can accommodate to in order to meet scalability requirements. Similarly, let q_i^{ft} be the number of cache instances each hardware switch $i \in \mathcal{H}$ should be connected to in order to meet fault-tolerance requirements. For maintaining correctness of shared state, popular SDN controller such as ONOS [13] and OpenDayLight suggest odd number of controller instances[1]. Accordingly, we also assume that q_i^{ft}

[1]ONOS and OpenDayLight use the RAFT consensus protocol which requires odd number of instances.

are odd. In particular, we choose q_i^{ft} from the set $\{1,3,5\}$ in random uniform fashion.

In order to reduce overall costs, our objective is to minimize the total number of cache instances required. This can be modeled as an integer programming problem as follows:

$$\begin{aligned}
\min \quad & \sum w(t)_j && \text{(1a)}\\
\text{subject to} \quad & \sum_{j\in\mathcal{C}} x(t)_{ij} = q_i^{ft}, \quad \forall i \in \mathcal{H} && \text{(1b)}\\
& \sum_{i\in\mathcal{H}} x(t)_{ij} \le q_j^s w(t)_j, \quad \forall j \in \mathcal{C} && \text{(1c)}\\
& x(t)_{ij} \le w(t)_j, \quad \forall i \in \mathcal{H}, \forall j \in \mathcal{C} && \text{(1d)}
\end{aligned}$$

where Equations (1b) and (1c) represent the fault-tolerance and scalability constraints. Equation (1d) introduces dummy variables $w(t)_j$ which are used to keep track of how many cache instances are utilized. We solve the formulated integer programming problem (1) using the GLPK solver[2].

Next, we consider the assignment problem between the hardware switches and the cache instances. Our objective is to reduce the overall cost of assignment. The cost associated with an assignment $x(t)$ comprises of two factors — a) the delay between the hardware switch i and cache instance j, and b) the control overhead in timeslot t.

A. Delay Model

The delay comprises of a) the propagation delay between the hardware switch i and cache instance j during multi-hop traversal using in-band communication, and b) the queuing delay at the cache instances, which depends on the load. We assume a load-balancing mechanism for flow-lookup in the software switches, which leads to uniformly distributed flow-lookup times (processing delay) among the software switches at all the cache instances. Therefore, we neglect the processing delay while making assignment decisions.

The propagation delay depends on the distance d_{ij} between the hardware switch i the cache instance j, and also depends on the communication media (coaxial cable, optical fiber, wireless). Mathematically, it is given as d_{ij}/v, where v is the propagation speed, which depends on the medium. For simplicity, we assume coaxial cable links with propagation speeds of $2.3 * 10^5$ km/sec [14]. The queuing delay at the cache instances depend on the incoming load (cache-miss) from the hardware switches. Let $\alpha(t)_i$ be the cache-miss rate of hardware switch i in pkts/sec in timeslot t. Therefore, the total arrival rate at cache instance j is given as $\lambda(t)_j = \sum_i \alpha(t)_i x(t)_{ij}$. Although arrivals from each individual switch i may be bursty, $\lambda(t)_j$ can be approximated as a Poission process using the Kleinrock independence approximation [15]. Therefore, we can model the arrival process at the cache instances using an M/M/1 queue, which results in a queuing delay of $1/(\mu_j - \lambda(t)_j)$, where μ_j represents the capacity of the instance in pkts/sec. Thus, the overall delay $\delta(t)$ associated with an assignment $x(t)$ is given as:

$$\delta(t) = \sum_i \sum_j \frac{d_{ij}}{v} + \sum_j \frac{1}{\mu_j - \lambda(t)_j} \tag{2}$$

B. Control Overhead

We consider in-band communication, where control messages are routed alongside the data-plane packets. Since bandwidth is limited, it is preferable to keep the control overhead low. Let h_{ij} denote the hop count between hardware switch i and cache instance j. Then, the overall control overhead in timeslot t is given as:

$$o(t) = \sum_j \sum_i h_{ij} \alpha(t)_i x(t)_{ij} \tag{3}$$

C. Rule-cache Assignment Problem

Our aim is to assign hardware switches to the cache instances to minimize the delay, while keeping the control overhead low. Therefore, using Equations (5) and (3), the assignment be modeled as an integer programming problem as follows:

$$\begin{aligned}
\min \quad & \eta\delta(t) + (1-\eta)o(t) && \text{(4a)}\\
\text{subject to} \quad & \lambda(t)_j \le \beta\mu_j, \quad \forall j \in \mathcal{C} && \text{(4b)}\\
& x(t)_{ij} \in \{0,1\}, \quad \forall i,j && \text{(4c)}\\
& \text{(1b)}, \text{(1c)}
\end{aligned}$$

Equation (4b) ensures that cache instances are not overloaded, where $\beta \in (0,1)$ represents a decay factor. The term $\eta \in [0,1]$ is a weight factor, used to signify the relative importance of delay and control overhead, which is dependent on the particular use-case. For instance, to improve resilience against control plane attacks such as denial-of-service, η can be lowered to assign more importance to network overhead. The integer programming problem (4) can be reduced to the generalized assignment problem, which is in most cases, NP-hard [16]. However, the solution to the assignment problem must be computationally efficient in order to be practical in time-varying network conditions. Therefore, we adopt a matching-theory based solution, which is considered an efficient approach for many networking problems [10].

IV. Solution Approach

In this Section, we model the traffic-aware rule cache assignment problem using a many-to-many matching framework in order to solve the problem efficiently.

A. Matching Concepts

We consider the hardware switches $\mathcal{H}$ and the cache instances $\mathcal{C}$ as two sets of players. The hardware switches' objective is to associate with the cache instance with the lowest delay $\delta(t)_{ij}$, while the cache instances prefer to associate with the hardware switches which contribute minimal control overhead $o(t)_{ij}$. According to matching theory, these

[2]https://www.gnu.org/software/glpk/

objectives are realized by specifying *preferences* of each player over subsets of the opposite set. For any hardware switch i, the notation $A \succ_i B$ is used to denote that i prefers set $A \subseteq \mathcal{C}$ over $B \subseteq \mathcal{C}$. A similar notation is used to represent the preferences of each cache instance. Given a potential set of cache instances $M_i \subseteq \mathcal{C}$, each hardware switch i can determine which subset of M_i it prefers to associate with. We use the notation $C_i(M_i)$ to denote this choice set.

Definition 1. *A many-to-many matching μ is a function that maps the set $\mathcal{H} \cup \mathcal{C}$ to the set of all subsets of $\mathcal{H} \cup \mathcal{C}$, and satisfies for all $i \in \mathcal{H}$ and $j \in \mathcal{C}$, the following [17]: i) $\mu(i) \in 2^{M_i}$, ii) $j \in \mu(i)$ iff $i \in \mu(j)$, iii) $|\mu(j)| \leq q^s_j$, and iv) $|\mu(i)| = q^{ft}_i$*

Let $A(i, \mu)$ denote the set of currently assigned partners of i under the matching μ. In other words, $A(i, \mu)$ denotes the set $j \in \mathcal{C}$ so that $i \in \mu(j)$ and $j \in \mu(i)$. For solving the rule cache assignment problem, we require a stable solution, so that there exists no players who prefer each other, but are not matched under the current assignment μ. In particular, we are concerned about *pairwise-stability*, which is defined as follows [17]:

Definition 2. *A matching μ is pairwise-stable if there exists no pair (i, j) with $i \notin \mu(j)$ and $j \notin \mu(i)$ such that if $T \in C_i(A(i, \mu) \cup \{j\})$ and $S \in C_j(A(j, \mu) \cup \{i\})$, then $T \succ_i A(i, \mu)$ and $S \succ_j A(j, \mu)$.*

B. Algorithm

The proposed algorithm is based on the deferred acceptance algorithm (DAA) [18] and proceeds in three stages — a) network statistics collection, b) preference relation specification, and c) two-sided matching, as presented in Algorithm 1. During the first phase, network information such as link latencies and flow-statistics are collected which is used to calculate network delay and controller overhead using Equations (5) and (3). Link latencies can be calculated analytically if distances are known beforehand, or may be measured in real-time using probing methods in SDN [19]. Similarly, control plane load can be inferred from monitoring SDN flow and packet-statistics[3]. In the second phase, the information collected in the first phase is used to specify the preference relations of the players, according to their goals (delay for hardware switches, and network overhead for cache instances). While building the preference relations for hardware switches, we observe from Equation (5) that the preferences are inter-dependent i.e., preferences for hardware switch $i \in \mathcal{H}$ is dependent on all other switches $k \in \mathcal{H} \mid k \neq i$. Roth *et al.* [18] show that with interdependent preferences, the solution requires computing preference relations over all subsets of $\mathcal{H}$ and $\mathcal{C}$, making the problem combinatorially complex. To overcome this problem, while computing $\delta(t)_{ij}$, we consider the worst case queuing delay at the cache instances. Mathematically,

[3]https://wiki.onosproject.org/display/ONOS/Control+Plane+Management+Application

$$\delta(t)_{ij} = \frac{d_{ij}}{v} + \sum_j \frac{1}{\mu_j - \beta_j \mu_j} \tag{5}$$

The third phase consists of two-sided matching between the hardware switches and cache instances, until pairwise-stability is achieved.

Algorithm 1 Dynamic Rule Cache Assignment

```
Inputs: Set of hardware switches $\mathcal{H}$ and set of cache instances $\mathcal{C}$ with associated requirements.
Output: A matching $\mu$ between $\mathcal{H}$ and $\mathcal{C}$.
   Phase 1 - Network statistics collection
1: Network information collected and used to calculate network delay and controller overhead.
   Phase 2 - Preference relation specification
2: Each $i \in \mathcal{H}$ and $j \in \mathcal{C}$ specifies its preference over the other set to build preference lists.
   Phase 3 - Two-sided matching
3: Each $j \in \mathcal{C}$ sends association request to its preferred subset $C_i(\mathcal{H})$.
4: Each $i \in \mathcal{H}$ refuses all expect the preferred $q^{ft}_i$ cache instances.
5: repeat
6:     Each $j \in \mathcal{C}$ sends association request to its preferred subset $C_i(\mathcal{H})$, including those already sent to who have not refused it yet.
7:     Each $i \in \mathcal{H}$ refuses all except the preferred $q^{ft}_i$ cache instances.
8: until convergence to a pairwise-stable outcome
```

C. Stability Analysis

Theorem 1. *Algorithm 1 always converges to a pairwise-stable matching between hardware switches $\mathcal{H}$, and cache instances $\mathcal{C}$.*

Proof. Let us assume the matching μ generated by Algorithm 1 is pairwise-unstable. Therefore, there exists a hardware switch $i \in \mathcal{H}$ and a cache instance $j \in \mathcal{C}$ with $i \notin \mu(j)$ and $j \notin \mu(i)$ such that for $T \in C_i(A(i, \mu) \cup \{j\})$ and $S \in C_j(A(j, \mu) \cup \{i\})$ we have $T \succ_i A(i, \mu)$ and $S \succ_j A(j, \mu)$. If $i \notin \mu(j)$ and $S \succ_j A(j, \mu)$, it implies $i \notin S$. However, $i \notin S$ leads to $i \notin C_j(A(j, \mu) \cup \{i\})$, which is a contradiction. Therefore, our assumption is wrong, and the matching μ is stable. ■

V. Performance Evaluation

For simulations, we consider the AttMpls topology from the Internet Topology Zoo [20], consisting of 25 switches (hardware) and 56 links. The link distances d_{ij} are calculated using the latitude and longitude information from the AttMpls topology. Solving the integer program (1), we obtain the minimum number of cache instances for this topology as $|\mathcal{C}| = 5$. Unless stated otherwise, we take scalability quota of cache instances as $q^s_j = 20$. The different parameters considered for the simulations are summarized in Table II.

To evaluate the effectiveness of the proposed dynamic rule-cache assignment scheme (TRC), we compare with a)

TABLE II: Simulation parameters

Parameter	Value
Number of hardware switches ($\|\mathcal{H}\|$)	25 [20]
Number of cache instances ($\|\mathcal{C}\|$)	5
Cache miss rate	$5-15\%$ [6]
Traffic rate	2000 flows/sec
Cache Agent capacity (μ_j)	$4000-5000$ pkts/sec
Propagation speed	$2.3*10^5$ km/sec [14]
Scalability quota (q_j^s)	20
Fault-tolerance quota (q_i^{ft})	$\{1,3,5\}$

random cache assignment (RCA), and b) minimum distance based cache assignment (MDC) [7].

Figure 3 shows the minimum number of cache instances required (as percentage of hardware switches) to meet scalability and fault-tolerance requirements. This is obtained by solving the integer programming problem 1 using the GLPK solver. From the figure, we note that on increasing the number of hardware switches, the number of cache instances required reaches a constant percentage (approximately 15%) of the hardware switches.

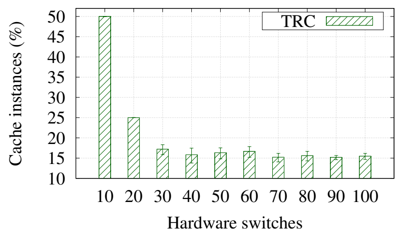


Fig. 3: cache instances vs hardware switches

Figure 4 presents the average switch to software cache delay with increasing traffic in the network. From the figure, we observe that the proposed scheme outperforms the MDC and RCA schemes in all cases. Overall, the proposed scheme, TRC, reduces the average delay by approximately 10% and 35% compared to the MDC and RCA schemes, respectively. The MDC scheme always associates a hardware switch with the nearest cache instance (if quota permits), which leads to load imbalance. This, in turn, leads to increased queuing delay due to which MDC incurs additional overall delay. On the other hand, the RCA scheme incurs significantly more delay due to sub-optimal choices which increase both propagation and queuing delay. However, it is evident from the figure that increase in traffic rate does not significantly increase the average delay. This implies that propagation delay, which is independent of traffic rate, dominates the queuing delay at the cache instances.

Figure 5 shows the total network overhead with increasing traffic in the network. From the figure, we observe that the proposed scheme outperforms the MDC and RCA schemes in all cases. Overall, the proposed scheme, TRC, reduces the network overhead by approximately 19% and 39% compared to the MDC and RCA schemes, respectively. In contrast to the delay results, we observe that increasing traffic rate significantly affects the total network overhead. In particular,

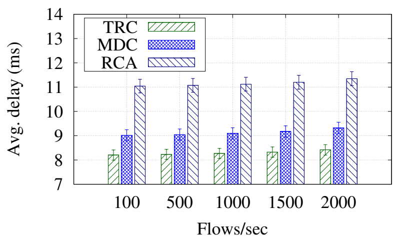


Fig. 4: Average delay with increasing flow rate

we observe that by taking into account the network overhead, the relative performance of the TRC scheme improves with increasing network traffic. Therefore, the proposed scheme is applicable to scenarios with bursty traffic, such as SDN for IoT applications [21], where it can dynamically re-assign cache instances to improve performance.

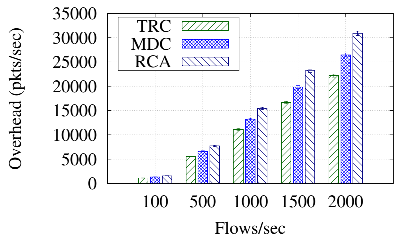


Fig. 5: Network overhead with increasing flow rate

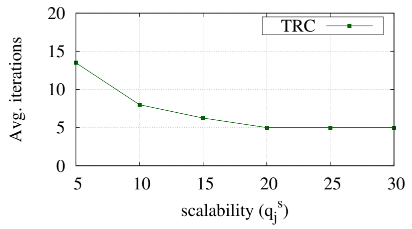


Fig. 6: Iteration times with $|\mathcal{C}|=5$

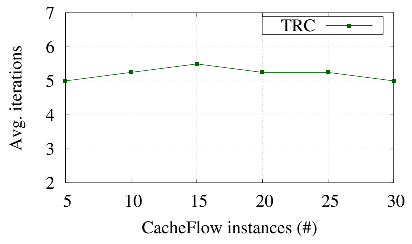


Fig. 7: Iteration times with $q^s=20$

Figures 6 and 7 show the iteration times of the proposed algorithm with varying values of scalability quota (q_j^s) and number of cache instances, respectively. From the figures, it is evident that the proposed algorithm is capable of quickly converging to a solution in a few iterations. Therefore, it can be used efficiently for dynamic re-assignment in varying network conditions. From Figure 6, we observe that with increasing scalability quota, iteration times decrease before becoming nearly constant. In particular, it is interesting to note that the iteration times converge to a value of 5, which is equal to the number of cache instances. Additionally, from Figure 7, we observe that the number of cache instances do not significantly affect iteration time of the proposed algorithm. However, $\mathcal{C} \leq 5$ makes the problem infeasible. Therefore, the integer problem (1) is able to minimize the number of software cache instances, thus reducing capital expenditure and energy costs of running extra cache instances, thereby making the proposed scheme sustainable.

VI. Security Implications

The centralized logic and rule-based management of SDN offer various advantages such as improved QoS and network programmability for applications such as Internet of Things [21], [22]. However, it introduces new security vulnerabilities particular to SDN, such as denial-of-service by attacking the centralized control plane or the limited flow-table [23]. The centralized control plane may be attacked by an adversary by sending packets intentionally crafted to trigger table-miss. On the other hand, an adversary may attack the flow-table by intentionally sending packets designed to install a large number of flow-rules, thereby causing flow-table overflow. Both of these attacks are assisted by the limited amount of memory in the hardware TCAM.

The proposed scheme mitigates these issues using a two-fold approach — a) considering a distributed software cache which includes redundancy for fault-tolerance, and b) by lowering the network overhead. The software-cache augmented architecture offers the advantages of a large flow space, which reduces the chances of table-miss. Further, the traffic-aware mapping scheme can dynamically switch the assignment of hardware switches to handle high load on one of the cache instances. This, in turn, leads to lower overall network overhead, as shown in Figure 5. The large flow-space offered by the software-cache also makes it harder for an adversary to cause a flow-table overflow attack. In this regard, the cache-based approach is superior to compression based approaches in terms of security, as reduces the chances of malicious flows passing undetected, as shown in Figure 1. Therefore, the distributed architecture and traffic-aware mapping of the proposed scheme ameliorates the effects of denial-of-service attacks in SDN.

VII. Conclusion

In this paper, we proposed a traffic-aware rule cache assignment strategy to improve overall performance in software defined networks. We formulated an integer programming model to minimize the number of software caches, while respecting scalability and fault-tolerance requirements. We modeled the cache assignment problem using a many-to-many matching framework, while considering the overall delay and network overhead. Simulation results show that the proposed scheme is capable of significantly reducing the overall delay as well as the total overhead in the network, which helps in mitigating denial-of-service security concerns in SDN.

References

[1] 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, 2008.

[2] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the 'One Big Switch' Abstraction in Software-defined Networks," in *Proc. of the ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, New York, USA, 2013, pp. 13–24.

[3] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "On the effect of forwarding table size on SDN network utilization," in *Proc. of the IEEE Conference on Computer Communications (INFOCOM)*, April 2014, pp. 1734–1742.

[4] M. Rifai, N. Huin, C. Caillouet, F. Giroire, D. Lopez-Pacheco, J. Moulierac, and G. Urvoy-Keller, "Too Many SDN Rules? Compress Them with MINNIE," in *Proc. of the IEEE GLOBECOM*, Dec. 2015, pp. 1–7.

[5] N. Saha, S. Bera, and S. Misra, "QoS-Aware Adaptive Flow-rule Aggregation in Software-Defined IoT," in *Proc. of the IEEE GLOBECOM*, 2018.

[6] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "CacheFlow: Dependency-Aware Rule-Caching for Software-Defined Networks," in *Proc. of the ACM SOSR*, New York, USA, 2016, pp. 6:1–6:12.

[7] A. Ruia, C. J. Casey, S. Saha, and A. Sprintson, "Flowcache: A cache-based approach for improving SDN scalability," in *Proc. of the IEEE INFOCOM Workshop*, April 2016, pp. 610–615.

[8] H. Li, S. Guo, C. Wu, and J. Li, "FDRC: Flow-driven rule caching optimization in software defined networking," in *Proc. of the IEEE ICC*, June 2015, pp. 5777–5782.

[9] H. Huang, S. Guo, P. Li, W. Liang, and A. Y. Zomaya, "Cost minimization for rule caching in software defined networking," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 4, pp. 1007–1016, April 2016.

[10] Z. Han, Y. Gu, and W. Saad, *Matching theory for wireless networks*. Springer, 2017.

[11] K. Hamidouche, W. Saad, and M. Debbah, "Many-to-many matching games for proactive social-caching in wireless small cell networks," in *Proc. of the IEEE International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, 2014, pp. 569–574.

[12] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic SDN controller assignment in data center networks: Stable matching with transfers," in *Proc. of the IEEE INFOCOM*, April 2016, pp. 1–9.

[13] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an Open, Distributed SDN OS," in *Proc. of the Workshop on ACM HotSDN*, 2014, pp. 1–6.

[14] S. Larsen, P. Sarangam, R. Huggahalli, and S. Kulkarni, "Architectural breakdown of end-to-end latency in a tcp/ip network," *International Journal of Parallel Programming*, vol. 37, no. 6, pp. 556–571, 2009.

[15] D. P. Bertsekas, R. G. Gallager, and P. Humblet, *Data networks*. Prentice-Hall International New Jersey, 1992, vol. 2.

[16] R. Cohen, L. Katzir, and D. Raz, "An efficient approximation for the generalized assignment problem," *Information Processing Letters*, vol. 100, no. 4, pp. 162–166, 2006.

[17] M. Sotomayor, "Three remarks on the many-to-many stable matching problem," *Mathematical social sciences*, vol. 38, no. 1, pp. 55–70, 1999.

[18] A. Roth and M. Sotomayor, *Two-Sided Matching: A Study in Game-Theoretic Modeling and Analysis*. Cambridge University Press, 1992.

[19] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, "OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks," in *Proc. IEEE NOMS*, 2014, pp. 1–8.

[20] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet Topology Zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, October 2011.

[21] N. Saha, S. Bera, and S. Misra, "Sway: Traffic-Aware QoS Routing in Software-Defined IoT," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2018, doi: 10.1109/TETC.2018.2847296.

[22] S. Bera, S. Misra, and A. V. Vasilakos, "Software-defined networking for internet of things: A survey," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1994–2008, Dec 2017.

[23] R. Kandoi and M. Antikainen, "Denial-of-service attacks in openflow sdn networks," in *Proc. of the IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 1322–1326.