Cost Minimization for Rule Caching in Software Defined Networking

Huawei Huang, *Student Member, IEEE,* Song Guo, *Senior Member, IEEE,* Peng Li, *Member, IEEE,* Weifa Liang, *Senior Member, IEEE,* and Albert Y. Zomaya, *Fellow, IEEE*

Abstract—Software-Defined Networking (SDN) is an emerging network paradigm that simplifies network management by decoupling the control plane and data plane, such that switches become simple data forwarding devices and network management is controlled by logically centralized servers. In SDN-enabled networks, network flow is managed by a set of associated rules that are maintained by switches in their local Ternary Content Addressable Memories (TCAMs) which support high-speed parallel lookup on wildcard patterns. Since TCAM is an expensive hardware and extremely power-hungry, each switch has only limited TCAM space and it is inefficient and even infeasible to maintain all rules at local switches. On the other hand, if we eliminate TCAM occupation by forwarding all packets to the centralized controller for processing, it results in a long delay and heavy processing burden on the controller. In this paper, we strive for the fine balance between rule caching and remote packet processing by formulating a minimum weighted flow provisioning (**MWFP**) problem with an objective of minimizing the total cost of TCAM occupation and remote packet processing. We show the problem is NP-hard, and propose an efficient offline algorithm if the network traffic is given, otherwise, we propose two online algorithms with guaranteed competitive ratios. Finally, we conduct extensive experiments by simulations using real network traffic traces. The simulation results demonstrate that our proposed algorithms can significantly reduce the total cost of remote controller processing and TCAM occupation, and the solutions obtained are nearly optimal.

1 Introduction

Software Defined Networking (SDN) is regarded as one promising next generation network architecture [1]–[3]. By shifting the control plane to a logically centralized controller, SDN offers programmable functions (e.g., OpenFlow [1], [4]) to dynamically control and manage packets forwarding and processing in switches, such that a wide range of network policies (e.g., traffic engineering [5]–[7], security [8], fault-tolerance [9]) and new network technologies can be easily deployed.

In SDN, each network flow is associated with a set of rules, such as packet forwarding, dropping and modifying, that should be installed at switches in terms of flow table entries along the flow path. SDN-enabled switches maintain flow rules in their local TCAMs [10]–[12], which support high-speed parallel lookup on wildcard patterns. A typically flow setup process [2], [13] between a pair of users, say users A and B, in SDN contains three steps. 1) User A sends out packets after connection initialization. Once a packet arrives a switch without matched flow table entries, this packet is forwarded to the controller. 2) Upon receiving the packet, the controller decides whether to allow or deny this flow according to network management policies. 3) If the flow is allowed, the controller installs corresponding rules to all switches along the path, such that consecutive packets can be processed by the installed rules locally at switches. Note that switches usually set an expiration time for rules, which defines the maximum rule maintenance time when no packet of associated flow arrives.

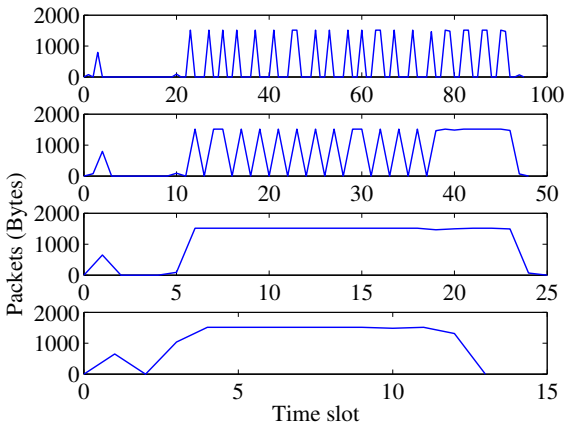


Fig. 1. The sniffed TCP traffic flow [14].

In practice, network flow shows various traffic patterns. For example, we show real-time traffic of four network flows [14] in Fig. 1, where some are burst transmission while the others have consecutive packet transmissions for a long time. For consecutive transmission, only the first packet experiences the delay of remote processing at the controller, and the rest will be processed by local rules at switches. However, for burst transmission, the corresponding rules cached in switches will be removed between two batches of packets if their interval is greater than the rule expiration time. As a

H. Huang, S. Guo and P. Li are with the School of Computer Science and Engineering, the University of Aizu, Japan. Email: {d8152101, sguo, pengli}@u-aizu.ac.jp
W. Liang is with the Research School of Computer Science, the Australian National University, Australia. Email: wliang@cs.anu.edu.au
A. Y. Zomaya is with the School of Information Technologies, the University of Sydney, Australia. Email: albert.zomaya@sydney.edu.au

result, remote packet processing would be incurred by the first packet of each batch, leading to a long delay and high processing burden on the controller. A simple method to reduce the overhead of remote processing is to cache rules at switches within the lifetime of network flow, ignoring the rule expiration time. Unfortunately, network devices are equipped with limited-space TCAMs because they are expensive hardware and extremely power-hungry. For instance, it is reported that TCAMs are 400 times more expensive [12] and 100 times more power-consuming [15] per Mbit than RAM-based storage. Since TCAM space is shared by multiple flow in networks, it is inefficient and even infeasible to maintain all rules at local switches. This dilemma motivates us to investigate efficient rule caching schemes for SDN to strive for a fine balance between network performance and TCAM usage.

The main contributions of our paper are summarized as follows.

- To the best of our knowledge, we are the first to study the rule caching problem with the objective of minimizing the sum of remote processing cost and TCAM occupation cost.
- We show the NP-hardness of the problem, and propose an offline algorithm by adopting a greedy strategy if the network traffic is given in advance. We also devise two online algorithms with guaranteed competitive ratios.
- Finally, we conduct extensive simulations using real network traffic traces to evaluate the performance of our proposals. The simulation results demonstrate that our proposed algorithms can significantly reduce the total cost of remote controller processing and TCAM occupation, and the solutions obtained are nearly optimal.

The reminder of the paper is structured as follows. Section 2 reviews some related work in table entries scheduling in SDN. Section III introduces the system model and problem formulation. An offline algorithm is proposed in Section IV. Two online algorithms are proposed and analysed in Section 5. Section 6 demonstrates the performance evaluation results. Finally, section 7 concludes this paper.

2 Related Work

As one pioneering work of recouping control plane from the data plane, NOX [8] has been proposed to control data forwarding based on OpenFlow.

Following this line of research, lots of efforts have been made on rule caching strategies in SDN, which can be classified into two categories: reactive way [2], [8] and proactive way [3], [4], [16].

The reactive rule caching has been widely adopted by existing work [2], [8] because of its efficient usage of TCAM space. The first packet of each "microflow" is forwarded to the controller that reactively installs flow entries in switches. For instance, Ethane [2] controller reactively installs flow table entries based on the first packet of each TCP/UDP flow. Recently, Bari et al. [13] use the on-demand approach to response flow setup requests.

On the other hand, other studies [17], [18] argue that reactive approach is time-consuming because of remote rule fetching, leading to heavy overhead in packet processing [5], [13], [16]. To reduce the response time for packets at switches without matched rules, proactive approach has been proposed to install rules in switches before corresponding packets arrive. For example, Benson, et al. [5] developed a system MicroTE that adapts to traffic fluctuations, with which rules can be dynamically updated in switches to imposes minimal overhead on network based on traffic prediction. Kang, et al. [3] have proposed to pre-compute backup rules for possible failures and cache them in switches in advance to reduce network recovery time.

In addition, other related literatures [7], [12], [19], [20] focus on the rules scheduling considering forwarding table size utilization. For instance, Katta et al. [12] proposed a abstraction of an infinite switch based on an architecture that leverages both hardware and software, in which rules caching space can be infinite. In that case, rules can be cached in forwarding table as many as possible. This abstraction saves TCAMs space, but the packet processing speed in switch is a bottleneck. To efficiently use TCAMs space, Kanizo et al. [7], Nguyen et al. [19] and Cohen et al. [20] propose their rules placement scheduling jointly consider the traffic routing in network. However, rules updating is ignored in their optimization. To the contrast, we study both the two aspects in our optimization.

The work most related with our paper is DIFANE [16], a compromised architecture that leverages a set of authority switches serving as a middle layer between the controller in control plane and switches in data plane. The endpoints rules are pre-computed and cached in authority switches. Once the first packet of a new microflow arrives the switch, the desired rules are reactively installed, from authority switches rather than the controller. In this way, the flow setup time can be significantly reduced. Unfortunately, caching pre-computed rules all in authority switches consumes large TCAM space. In our work, we still load the flow rules into switches in a reactive way. However, rule caching period is controlled by our proposed algorithm by taking both remote processing and TCAM occupation cost into consideration.

3 System model and Problem Formulation

We consider a discrete time model, where the time horizon is divided into T time slots of equal length l. A network flow travels along a set of SDN-enabled switches, each of which is assigned a set of rules to implement routing or network management functions.

TABLE 1
Notions and Variables

Notations	Description
T	maximum time-slot range of consideration
l	length of each time slot (seconds)
a_i	a binary indicator that denotes the positive flow rate in time slot i, i=1,2,..T
δ	sum of time slots where $a_i > 0$
D	amount of valid periods during [1, T]
V_i	the i^{th} valid period during [1, T]
E_i	the i^{th} empty period during [1, T]
α	occupation cost of caching table entry
β	remote processing cost
γ	$\gamma = \frac{\alpha}{\beta}$
x_i	a binary variable indicating whether cache action happens in time slot i, i=1,2,..T
y_i	a binary variable indicating whether fetch action happens in time slot i, i=1,2,..T
C_C	total cost of cache action during [1, T]
C_F	total cost of fetch action during [1, T]

For simplicity, in this paper, we study rule caching at a switch with a set of associated rules whose maintenance cost is α per second, and our results can be directly extended to all switches along the flow path. Note that the set of rules associated with the network flow will be cached or removed as a whole at the switch. When no matched flow table entries are available at the switch for arriving packets, these packets will be sent to the controller for processing. We model the remote processing cost at controller as β. The ratio between these two kinds of cost is denoted by γ, i.e., $\gamma = \frac{\alpha}{\beta}$. We consider arbitrary traffic pattern of the network flow. The set of time slots with ($a_t > 0$) and without ($a_t = 0$) packet transmission are referred to as valid period and empty period, respectively. All symbols and variables used in this paper are summarized in Table 1.

We define a binary variable x_i to denote whether a flow entry is cached at the i-th time slot:

$$x_i = \begin{cases} 1, & \text{if rules are cached at the } i\text{-th time slot,} \\ 0, & \text{otherwise.} \end{cases}$$

The occupation cost can be calculated as follows.

$$C_C = \alpha l \cdot \sum_{i=1}^{T} x_i. \tag{1}$$

We also define a binary variable y_i to indicate whether remote packet processing is conducted.

$$y_i = \begin{cases} 1, & \text{if rules are remotely fetched in the } i\text{-th time slot,} \\ 0, & \text{otherwise.} \end{cases}$$

and the corresponding cost can be calculated by:

$$C_F = \beta l \cdot \sum_{i=1}^{T} y_i. \tag{2}$$

With the global information of the given traffic flow, the minimum weighted flow provisioning (**MWFP**) problem can be formulated as follows:

$$\begin{aligned} \textbf{MWFP}: \ & \min_{x_i, y_i} C_{Total} = C_C + C_F \\ \text{s.t.} \quad & x_i - x_{i-1} \le y_i, && \text{(3a)} \\ & \sum_{j=1}^{i} y_j \ge x_i, && \text{(3b)} \\ & (x_i + y_i) \ge a_i, && \text{(3c)} \\ & x_i, y_i \in \{1, 0\}, \forall i = 1, 2, .., T. \end{aligned}$$

The trigger of remote processing is represented by constraint (3a), where $y_i = 1$ only when we decide to cache rules from the i-th time slot, which are not available at local switches in the $(i-1)$-th time slot, i.e., $x_i = 1$ and $x_{i-1} = 0$. Constraint (3b) indicates that the switch needs to fetch rules at least one time before caching. Constraint (3c) claims that arriving packets must be processed by local cached rules or remote controller. Note that the input of this problem are a_i, α, β or γ, and the output are the scheduling solution x_i, y_i, i=1,2,..T.

Theorem 1. *The **MWFP** problem is NP-hard.*

Proof: We show **MWFP** problem is NP-hard by performing a polynomial reduction from the weighted set cover problem, a classic combinatorial optimization problem that is proved NP-hard [21], [22]: Suppose we have an instance of the weighted set cover problem $\mathcal{A}'$=($\mathcal{U}'$,$\mathcal{S}'$). Given a set $\mathcal{U}' = \{1, 2, .., T\}$, a collection $\mathcal{S}'$=$\{S'_1, S'_2, ..., S'_m\}$($m$=$\frac{1}{2}(T^2 + T)$) of subsets of $\mathcal{U}'$, with weights $w(S'_i)$, the goal is to find a collection $\mathcal{C}'$ of these subsets whose union is $\mathcal{U}'$, such that $\sum_{C'_i \in \mathcal{C}'} w(C'_i)$ is minimized.

We construct an instance of the **MWFP** from this instance, i.e., $\mathcal{A}$= ($\mathcal{U}$, $\mathcal{S}$) with $\mathcal{U} = \{1, 2, .., T\}$, $\mathcal{S}$ =$\{$ {1},{2},.., {T}, {1,2},{2,3},.., {T-1,T}, {1,2,3},{2,3,4},.., {T-2,T-1,T}, ..., {1,2,..,T-1}, {2,3,..,T}, {1,2,..,T}$\}$, in which each element in each subset of S denotes the time-slot during [1,T]. Every subset in S is a *cost*(S_i), which can be calculated as

$$\textit{cost}(S_i) = \begin{cases} \beta l + \alpha l \cdot |S_i|, & \text{if } |S_i| > 1; \\ \beta l \cdot a_i, & \text{if } |S_i| = 1. \end{cases} \tag{4}$$

The target is to find a collection of time slots $\mathcal{C} \subseteq \mathcal{S}$ such that $\bigcup_{C_i \in \mathcal{C}} C_i = \mathcal{U}$, and ensuring the total cost of $\mathcal{C}$ is minimum. It can be seen that such transformation can be done in polynomial time. As the weighted set cover problem is NP-hard, the **MWFP** is NP-hard too as the reduction can be done in polynomial time, and vice versa. □

4 OFFLINE ALGORITHM

Due to the NP-hardness, solving **MWFP** problem is computationally infeasible for large-scale problem instances. In this section, we propose a heuristic algorithm to solve the offline version of the problem. As shown in Algorithm Offline Greedy, we first generate a collection of time slot sets that represents possible rule cache periods as

shown in line 1. Each set is assigned a weight according to (4) in line 2. Then, we select time slot sets for rule caching in an iterative manner from line 4 to 8. In each iteration, we choose the set X that minimizes the value of weight $w(X)$ divided by number of elements not yet covered.

Algorithm 1 Offline Greedy Algorithm to solve **MWFP**

Input: flow indicator set $\mathcal{F} = \{a(t), t\in[1,T]\}$, and $\mathcal{T}=\{1,2,...,T\}$

Output: The collection C of subsets of $\mathcal{T}$

1: generate sample collection $\mathcal{S}$ with $\bigcup_{S\in\mathcal{S}} S = \mathcal{T}$, where its generating method has been described in the proof of Theorem 1.
2: generate the weight function w: $\mathcal{S} \to \mathbb{R}_+$ by invoking (4)
3: $C \leftarrow \emptyset$, and $R \leftarrow \mathcal{T}$
4: **while** $R \neq \emptyset$ **do**
5: $\quad X \leftarrow \arg\min_{X\in\mathcal{S}} \frac{w(X)}{|X\cap R|}$
6: $\quad C \leftarrow C \cup X$
7: $\quad R \leftarrow R \setminus X$
8: **end while**

Theorem 2. *Alg.1 is ($\ln T$+1)-approximation to the optimal solution of **MWFP**, where T is the maximum time slot.*

Proof: For each element (time slot) $t_j \in \mathcal{T}$, let S_j be the first picked set that covers it while applying Alg.1, and $\theta(t_j)$ denote the amortized cost of each element in S_j,

$$\theta(t_j) = \frac{w(S_j)}{|S_j \cap R|}.$$

Obviously, the cost of Alg.1 can be written as $\sum_{t_j\in\mathcal{T}} \theta(t_j)$.

Then, let $\widehat{\mathcal{T}} = \{t_1, t_2, ..., t_T\}$ denote the ordered set of elements in [1,T] that each is covered. Note that, when t_j is to be covered, apparently we have $R \supseteq \{t_j, t_{j+1}, ..., t_T\}$. We can see that R contains at least $(T - j + 1)$ elements. Therefore, the amortized cost in S_j is at most the average cost of the optimum solution (denoted by $\texttt{OPT}$), i.e.,

$$\theta(t_j) = \frac{w(S_j)}{|S_j \cap R|} \leq \frac{\texttt{OPT}}{T - j + 1}.$$

By summing the $\theta(t_j)$ in all time slots, we get:

$$\sum_{t_j\in\mathcal{T}} \theta(t_j) \leq \texttt{OPT}(\frac{1}{T} + \frac{1}{T-1} + \cdots + \frac{1}{2} + 1).$$

That is $\texttt{Greedy} \leq \texttt{OPT} \cdot H(T) \leq \texttt{OPT} \cdot (1 + \ln T)$, where $H(T)$ is called the harmonic number of T. □

5 Online Algorithms

In this section, we consider the **MWFP** problem assuming that the packet traffic information is not given in advance. We first present several important observations in the optimal solution, followed by two proposed online algorithms with low computational complexity to approximate the optimal solution.

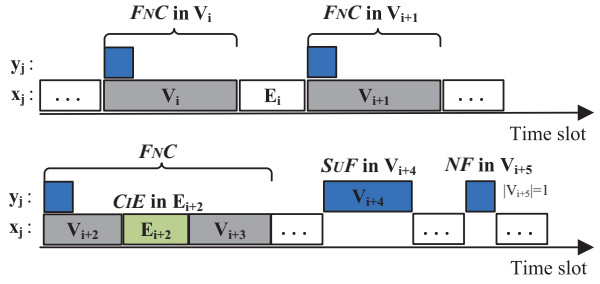


Fig. 2. Illustration of typical actions in optimum solution.

5.1 Typical actions in optimal solutions

By carefully examining the optimal solutions of several problem instances, we find that there exists several typical actions as follows.

- FNC (Fetch and Cache): for valid periods with at least 2 time slots, flow rules are first fetched from the remote controller, and then they are cached at local switches.
- SUF (Successive Fetch): for a valid period with at least 2 time slots, all packets are forwarded to the remote controller for processing.
- CIE (Cache in Empty period): rules are cached in switches during the empty period between two valid periods.
- NF (Only Forward): Packets are processed by the controller in the empty periods of one time slot.

Above typical actions are illustrated in Fig. 2. The optimal solutions can be categorized into following three cases.

- **OPT-A**: there are only SUF actions in the optimal solution.
- **OPT-B**: there are only CIE actions in the optimal solution.
- **OPT-C**: both SUF and CIE actions exist in the optimal solution.

5.2 Online Exactly Match the Flow Algorithm

Our first online algorithm, which is referred to as EMF (Exactly Match the Flow Algorithm), is shown in Alg. 2. In each time slot, each switch makes a decision, caching or fetching, according to observed network traffic. When there are packets arriving in the current time slot, i.e., $a_t = 1$, if no matched rules are cached, i.e., $x_{t-1} = 0$, switch fetches rules from the controller. Otherwise, we keep caching them in switches.

Lemma 1. *Suppose there are D valid periods (denoted by V_i, i = 1,2,...,D) including δ valid time slots within $[1, T]$, the total cost of* EMF *is*

$$C_{EMF} = \beta l D + \alpha l \delta. \quad (5)$$

Algorithm 2 online Exactly Match the Flow (EMF) Algorithm

```
for each time slot t ∈ [1, T] do
  if a_t=1 and x_{t-1}=0 then
    fetch flow rules from the controller
    y_t ← 1, x_t ← 1
  else if a_t=1 and x_{t-1}=1 then
    keep caching entries in switches
  else if a_t=0 and x_{t-1}=1 then
    remove the corresponding flow table entries
    x_t ← 0
  end if
end for
```

Proof: In Alg. 2, flow rules are maintained in switches only when there are network traffic passing through, such that TCAM occupation cost can be easily calculated by $\alpha l\delta$. Since rule fetching action happens in the beginning of each valid period, we have fetching cost of βlD. By summing them up, the total cost of EMF can be calculated by (5). □

Lemma 2. *When the optimal solution belongs to OPT-A,* SUF *is adopted by any valid period V_i, we have $\gamma > \frac{|V_i|-1}{|V_i|}, \forall i = 1, 2, ..., D$.*

Proof: Since SUF is adopted in the optimal solution, its cost must be less than FNC, i.e.,

$$|V_i|\beta l < \beta l + |V_i|\alpha l$$
$$\Rightarrow \beta < \frac{|V_i|\alpha}{|V_i|-1} \Rightarrow \gamma > \frac{|V_i|-1}{|V_i|}, \forall i = 1, 2, ..., D.$$

□

Theorem 3. *When the optimal solution belongs to OPT-A, the* EMF *algorithm is $(\gamma + \frac{D}{\delta})$-competitive.*

Proof: Since the optimal solution belongs to OPT-A, i.e., all packets are sent to the controller for processing, it is easy to see that the total cost of optimal solution is $\beta l\delta$. Thus, the competitive ratio λ_A is:

$$\widehat{\lambda}_A = \frac{C_{\text{EMF}}}{\beta l\delta} = \frac{\alpha\delta + \beta D}{\beta\delta} = \gamma + \frac{D}{\delta}.$$

□

Lemma 3. *When the optimal solution belongs to OPT-B, rules are always maintained by switches once downloaded, i.e.,* CIE *is adopted by the period E_i between any two valid periods V_i and V_{i+1}, and we have $\gamma < \frac{1}{|E_i|}, \forall i = 1, 2, ..., D-1$.*

Proof: If switches continue to cache rules once they're downloaded, the TCAM occupation cost during E_i must be less than the fetching cost at the beginning of V_{i+1}, i.e.,

$$\alpha l(|E_i| + |V_{i+1}|) < \beta l + |V_{i+1}| \cdot \alpha l$$
$$\Rightarrow \beta > \alpha|E_i| \Rightarrow \gamma < \frac{1}{|E_i|}, \forall i = 1, 2, ..., D-1.$$

□

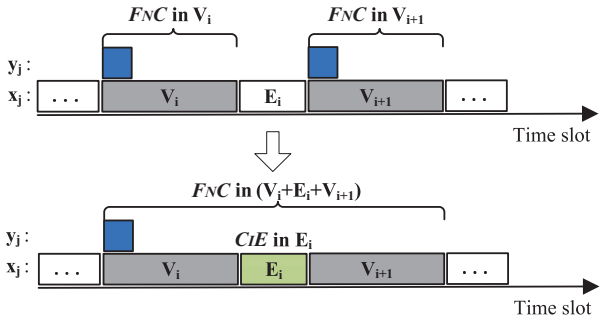


Fig. 3. Rules are cached in E_i because of CIE action.

Theorem 4. *When the optimal solution belongs to OPT-B, the* EMF *algorithm is $(\frac{D+\gamma\delta}{1+\gamma(\delta+D-1)})$-competitive.*

Proof: As shown in Fig. 3, since rules are maintained in switches once downloaded under OPT-B, its total cost can be easily calculated by $\beta l + \alpha l(\delta + \sum_{i=1}^{D-1}|E_i|)$, where the first term is the cost of the only fetching that happens in the beginning of the first valid period, and the second term is TCAM occupation cost. Combined with Lemma 1, the competitive ratio is:

$$\widehat{\lambda}_B = \frac{C_{\text{EMF}}}{\beta l + \alpha l(\delta + \sum_{i=1}^{D-1}|E_i|)} \leq \frac{\beta D + \alpha\delta}{\beta + \alpha(\delta + D - 1)} = \frac{D + \gamma\delta}{1 + \gamma(\delta + D - 1)}.$$

□

Lemma 4. *When the optimal solution belongs to OPT-C, there exists at least one valid period $V_i, i \in [1, D]$ and one empty period $E_j, j \in [1, D-1]$, such that $\frac{|V_i|-1}{|V_i|} < \gamma < \frac{1}{|E_j|}$.*

Proof: This lemma can be easily proved following similar analysis in Lemmas 2 and 3. □

Theorem 5. *When the optimal solution belongs to OPT-C, the* EMF *algorithm is $(\frac{D+\gamma\delta}{D+\gamma(\delta-D+2)})$-competitive.*

Proof: Without loss of generality, we suppose CIE is adopted in empty periods $\{E_1, E_2, ..., E_x\}$, and SUF is adopted in valid periods $\{V_1, V_2, ..., V_y\}$. There are z NF actions in the optimal solution. The total cost of optimal solution belongs to OPT-C can be calculated by:

$$\begin{aligned} C_{OPT-C} &= \beta l[D - x + \sum_{i=1}^{y}(|V_i| - 1)] + \\ &\quad \alpha l(\delta + \sum_{j=1}^{x}|E_j| - \sum_{i=1}^{y}|V_i|) - \alpha lz, \\ &= \beta lD + \alpha l\delta + h(x, y, z), \end{aligned}$$

where

$$\begin{aligned} h(x, y, z) &= \alpha l\sum_{j=1}^{x}|E_j| - \beta lx + [\beta l\sum_{i=1}^{y}(|V_i| - 1) - \\ &\quad \alpha l\sum_{i=1}^{y}|V_i|] - \alpha lz. \end{aligned}$$

Referring to **Lemma 4**, $\gamma < \frac{1}{|E_j|} \Rightarrow \alpha - \beta < 0$ and $\frac{|V_i|-1}{|V_i|} < \gamma \Rightarrow \beta l \sum_{i=1}^{y}(|V_i|-1) - \alpha l \sum_{i=1}^{y}|V_i| < 0$. Therefore, $h(x,y,z) < 0$ and the ratio $\lambda_C > 1$.

Obviously, we have $x \geq 1$, $y \geq 1$ and $z \geq 0$. We then consider two extreme cases. In the first case, there exists multiple CIE actions but only one SUF action. Since there must be one empty period between these two types of patterns, CIE actions cover at most $D-2$ empty periods, i.e., $x \leq D-2$. In the second case, there are only one CIE action and multiple SUF actions. The only one CIE occupies at least two valid periods. As a result, SUF actions use at most $D-2$ valid periods, i.e., $y \leq D-2$. Furthermore, the constitution of y SUF actions occupy y valid periods, and the other $D-y$ ones are left to CIE and NF actions, which cover x empty periods and z tiny valid periods, respectively. If there is only one CIE pattern, in which all the $D-y$ valid periods are crossed with x empty periods, we have $x_{max}+1+z=D-y$, i.e., $x+y+z \leq D-1$. Finally, we obtain a feasible region of $h(x,y,z)$, which is denoted by $\Lambda = \{(x,y,z)|1 \leq x \leq D-2, 1 \leq y \leq D-2, z \geq 0, x+y+z \leq D-1\}$, where x, y and z are all integers. The lower bound of $h(x,y,z)$ is derived as follows.

$$\begin{aligned} h(x,y,z) &= \alpha l \sum_{j=1}^{x}|E_j| - \beta l x + (\beta-\alpha) l \sum_{i=1}^{y}|V_i| - \\ &\quad \beta l y - \alpha l z, \\ &\geq (\alpha-\beta) l x + (\beta-\alpha) l \cdot 2y - \beta l y - \alpha l z, \\ &= (\alpha-\beta) l x + (\beta-2\alpha) l y - \alpha l z, \\ &\geq (\alpha-\beta) l + (\beta-2\alpha) l - \alpha l (D-3), \\ &= \alpha l (2-D). \end{aligned}$$

Therefore, the competitive ratio can be expressed by:

$$\lambda_C \leq \frac{\beta D + \alpha\delta}{\beta D + \alpha\delta + \alpha(2-D)} = \frac{D+\gamma\delta}{D+\gamma(\delta-D+2)}.$$

□

5.3 Online Extra η time-slot Caching Algorithm

Our proposed EMF algorithm attempts to minimize the TCAM occupation cost by caching flow rules only when there are network traffic passing through switches. However, it would incur frequent remote processing at the controller under burst packet transmissions. In this subsection, we study to further reduce total cost by proposing the ECA (Extra Cache Algorithm (ECA) algorithm that specifies an expiration time for cached rules. As shown in Alg. 3, we specify a parameter η as input. In each time slot, if we decide to conduct fetching action, the expiration time of fetched rules, which is denoted by `idle_timeout`, is set to ηl. We show how to set the value of η to achieve the closest performance with optimal solution by empirical analysis in next section.

Algorithm 3 online Extra Cache Algorithm (ECA)

Input: η

1: **for** each time slot $t \in [1,T]$ **do**
2: **if** fetch action happens in slot t **then**
3: `idle_timeout` $\leftarrow \eta l$ for all entries to be installed
4: **end if**
5: t++
6: **end for**

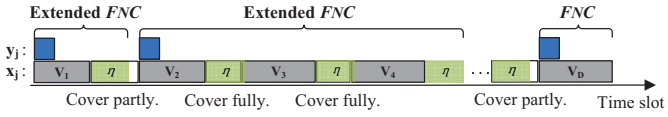


Fig. 4. An example of the ECA algorithm.

5.3.1 General cases of ECA

We let N_0 denote the number of empty periods whose length is less than η. By representing the total number of empty time slots covered by η with L, we have the following theorem.

Theorem 6. *The competitive ratios of* ECA *over* ***OPT-A***, ***OPT-B*** *and* ***OPT-C*** *are* $\zeta_A = \frac{D-N_0+\gamma(\delta+L)}{\delta}$, $\zeta_B = \frac{D-N_0+\gamma(\delta+L)}{1+\gamma(\delta+D-1)}$ *and* $\zeta_C = \frac{D-N_0+\gamma(\delta+L)}{D+\gamma(\delta-D+2)}$, *respectively.*

Proof: As shown in Fig. 4, if the length of an empty period is longer than η, cached rules will be removed after the expiration time, and they will be refetched at the beginning of next valid period. The total TCAM occupation cost can be calculated by $\alpha l(\delta+L)$. Otherwise, rules are cached at the switch during the empty period, leading to remote processing cost $\beta l(D-N_0)$. Therefore, the total cost of ECA with η is

$$C_{ECA(\eta)} = \beta l(D-N_0) + \alpha l(\delta+L). \tag{6}$$

Following the similar analysis in Theorems 3, 4, and 5, we can easily obtain the competitive ratios of $\zeta_A = \frac{D-N_0+\gamma(\delta+L)}{\delta}$, $\zeta_B = \frac{D-N_0+\gamma(\delta+L)}{1+\gamma(\delta+D-1)}$ and $\zeta_C = \frac{D-N_0+\gamma(\delta+L)}{D+\gamma(\delta-D+2)}$ over OPT-A, OPT-B, and OPT-C, respectively. □

5.3.2 Special case of ECA

Suppose the length (denoted by variable X) of all the empty periods E_j $(j = 1,2,...,D-1)$ are exponentially distribution with mean value μ_e, i.e., $X \sim \exp(\frac{1}{\mu_e})$, the value of N_0 can be calculated by:

$$N_0 = (D-1) \cdot Pr(X \leq \eta l) = (D-1)(1-e^{\frac{-\eta l}{\mu_e}}). \tag{7}$$

And the total length of the completely covered empty periods by η can be written as

$$\begin{aligned} L_1 &= (D-1) \cdot E(X \leq \eta l) \\ &= (D-1) \int_0^{\eta l} (X(\frac{1}{\mu_e} e^{\frac{X}{-\mu_e}}))dX \\ &= \mu_e (D-1)[1-(\frac{\eta l}{\mu_e}+1)e^{\frac{-\eta l}{\mu_e}}]. \end{aligned} \tag{8}$$

On the other hand, the total length in the empty periods where partially cut by η shall be simply calculated as

$$L_2 = (D - 1 - N_0) \cdot (\eta l) = \eta l (D-1) e^{\frac{-\eta l}{\mu_e}}. \quad (9)$$

Therefore, the cost of ECA with parameter η, μ_e and l shall be

$$\begin{aligned} C_{ECA(\eta,\mu_e,l)} &= \alpha l(\delta + L_1 + L_2) + \beta l(D - N_0) \\ &= \alpha(\delta l + (D-1)\mu_e(1 - e^{\frac{-\eta l}{\mu_e}})) + \beta l(1 + (1-D)e^{\frac{-\eta l}{\mu_e}}). \end{aligned} \quad (10)$$

Similarly, the competitive ratio of ECA under this special case can also be derived following the approach in the proof of Theorem 6. In performance evaluation, we conduct extensive simulations under various network settings to find out the best value of η leading to closet performance with optimal solution.

6 Evaluation

We conduct extensive simulations in this section to evaluate the performance of our proposed algorithms and the derived competitive ratios. We adopt network traces from [14], which contain TCP traffics captured at a wired sniffer. Each trace file is collected within around 50 seconds. Furthermore, the offline optimal solutions (denoted by Offline OPT) are obtained using commercial solver Gurobi optimizer [23]. In each set of simulation, we always fix α=10, the settings of other parameters are labeled below the figures.

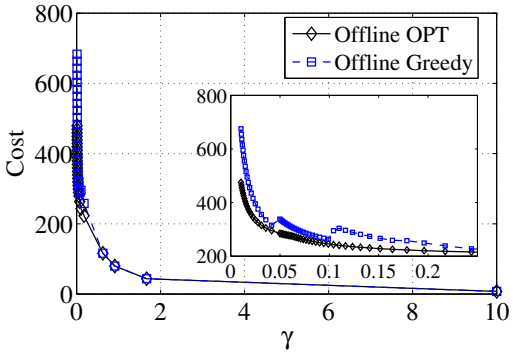
6.1 Evaluation of Offline Algorithm

As shown in Fig. 5(a), when $\alpha = 10$ and $l = 0.25$ s, total cost of both algorithms decreases as γ grows from 0.01 to 10. This is because under larger γ, fetching is preferred as it leads to low cost. Specifically, we observe that two algorithms converge when γ is greater than 1 because they generate only fetch operations.

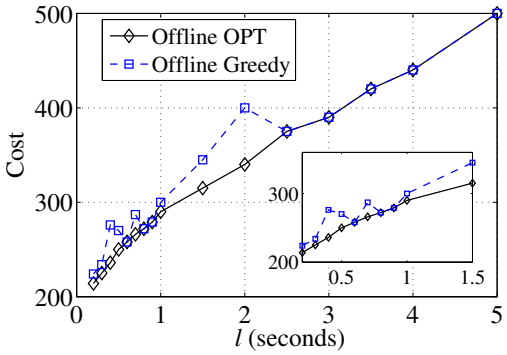
We then investigate the influence of time slot length on the total cost by changing the value of l from 0.1s to 5s. As shown in Fig. 5(b), total cost increases as the growth of l under optimal solutions. We observe that some fluctuation in the performance of our greedy algorithm. That is because in each iteration of while loop in Alg.1, we prefer tiny valid periods, which may some time slots already contained. The performance ratio of our algorithm and the optimal solution is shown in Fig. 5(c), where the ratio is very close to 1, much better than the analytical result, the upper bound.

6.2 Evaluation of Online EMF and ECA

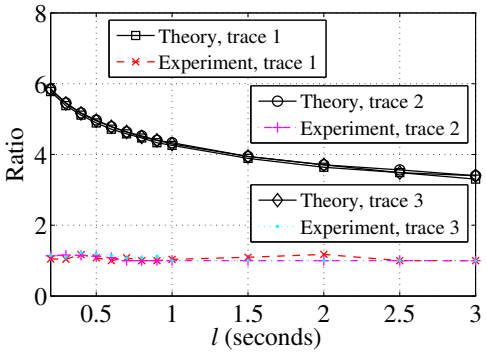
Then, the online EMF and ECA algorithms are evaluated with real network traffic traces under three cases, respectively.



(a) l=0.25 s, Trace 1



(b) γ=0.2, Trace 1

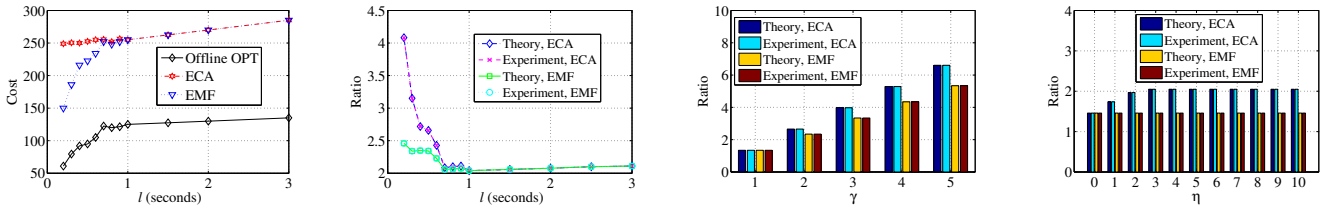


(c) γ=0.2

Fig. 5. Performance of offline Algorithm 1.

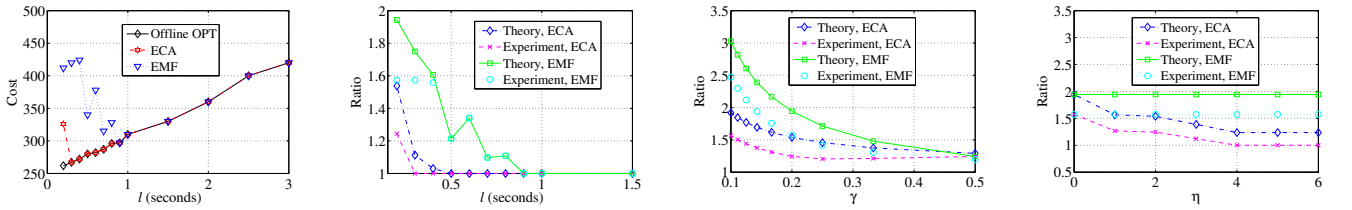
6.2.1 Over OPT-A

The total cost under different values of l is shown in Fig. 6(a), where results of all algorithms show as increasing functions of l. On the other hand, their performance ratio with the optimal solutions decreases as the growth of l as shown in Fig. 6(b). Furthermore, we observe that the performance ratio are always equal to the derived upper bound. That is because only SUF and NF patterns exist in the optimal solutions under this case, i.e., packets are always processed by the remote controller. Finally, we show the performance ratio under different value of η in Fig. 6(d). We observe that ratio of ECA is same with EMF when η=0, and the ratios of EMF never change because of fixed γ and l. In Figs. 6(b), 6(c), and 6(d), EMF outperforms ECA with closer performance to the



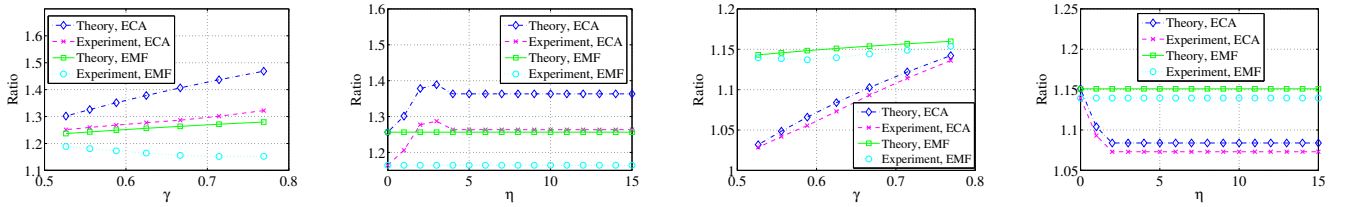
(a) γ=2, η=5, trace 3 (b) γ=2, η=5, trace 3 (c) η=5, l = 0.5 s, trace 3 (d) γ=1, l = 0.2 s, trace 3

Fig. 6. Performance of EMF and ECA over OPT-A.



(a) γ=0.2, η=2, trace 2 (b) γ=0.2, η=2, trace 2 (c) η=2, l = 0.2 s, trace 2 (d) γ=0.2, l = 0.2 s, trace 2

Fig. 7. Performance of EMF and ECA over OPT-B.



(a) η=2,l=0.2 s,trace 2 (b) γ=0.625,l=0.2 s,trace 2 (c) η=2,l=0.4 s,trace 2 (d) γ=0.625,l=0.4 s,trace 2

Fig. 8. Performance of EMF and ECA over OPT-C.

optimal solution under most of settings. We attribute this phenomenon to the fact that rules are cached at switches for a longer time under ECA.

6.2.2 Over OPT-B

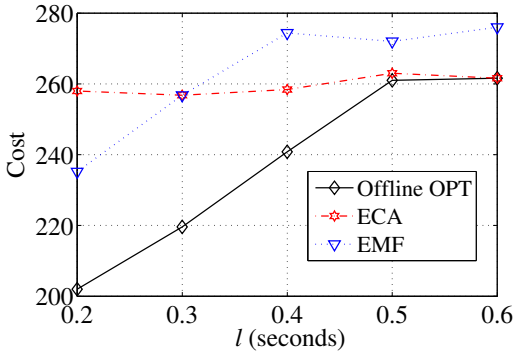
In this case, packets tend to be processed at local switches because γ becomes small. In Figs. 7(a) and 7(b), we have similar observations with the results under OPT-A case, except ECA outperforms EMF. This can be attributed to that there are mainly CIE patterns and few NF patterns under OPT-B case, and the extra caching durations of ECA cover many empty periods. Accordingly, the cost of ECA is smaller than EMF, particularly when γ and η become large. Finally, both algorithms converge to the optimal solution under larger value of l. In respect of performance ratio, Fig. 7(c) shows ratio is decreasing function of γ for both ECA and EMF. Because larger γ leads to more short FCN patterns in optimal solution, which makes ECA and EMF close to optimal solution. Therefore, their ratios decrease and approach to 1 gradually. As we mentioned above, Fig. 7(d) shows the benefit of a larger η in ECA, because more empty periods are covered and much fetching cost can be saved under OPT-B case.

6.2.3 Over OPT-C

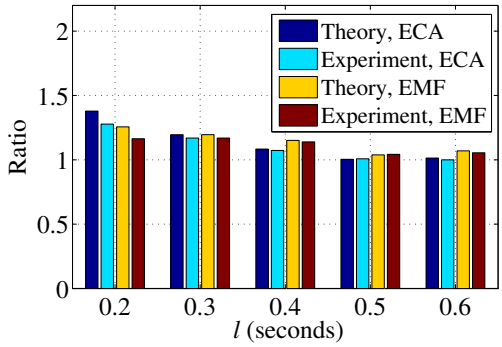
The performance of ECA and EMF is investigated in Figs. 8 and 9. We observe that ECA and EMF show distinct performance under different settings. For example, in Fig. 9(a), the cost of ECA is larger than EMF when l=0.2, but it becomes opposite when l is greater than 0.3. We have similar observations in other figures. Interestingly, in Fig. 8(b), we observe that the curve of ECA first increases, and then decreases when η is greater than 3, finally converging to 1.28 after $\eta = 4$. This is because when η is small, it only covers few empty periods with short length, and the advantage of extra caching is not obvious. As η becomes larger, the number of covered empty periods grows, leading to reduced total cost. When all empty periods are covered by large η, the performance of ECA becomes stable. In Fig. 8(d), the ratios of ECA keep decreasing and then converging because short empty periods become fewer when l increases to 0.4.

6.3 Evaluation of Special Case of ECA

Finally, we study the performance of ECA when lengths of empty periods are exponential distribution. We consider randomly generated network traffic with μ_e=1.0 and T=100.



(a) γ=0.625, η=2, trace 2



(b) γ=0.625, η=2, trace 2

Fig. 9. Performance of EMF and ECA over OPT-C while varying l.

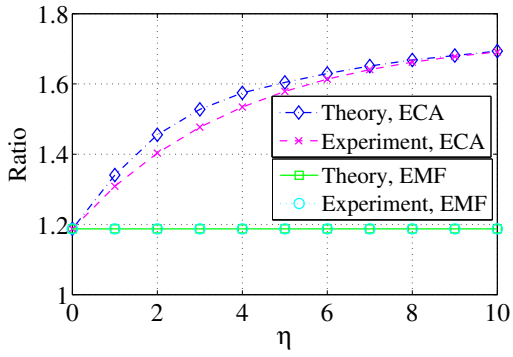


Fig. 10. ECA over OPT-A under special case, γ=1, μ_e=1.0, l=0.25.

As shown in Fig. 10, ratio of ECA algorithm increases as η grows from 0 to 10, which shows the same performance with Fig. 6(d). In Fig. 11, we have similar observation with Fig. 8(b) because of the same reasons. In Fig. 12, we set l=0.5, and performance ratios are always below the theoretical bound as γ and η changes within [0.5,0.8] and [0,10], respectively. The simulation results also suggest that η shall be set to small values no matter how γ changes.

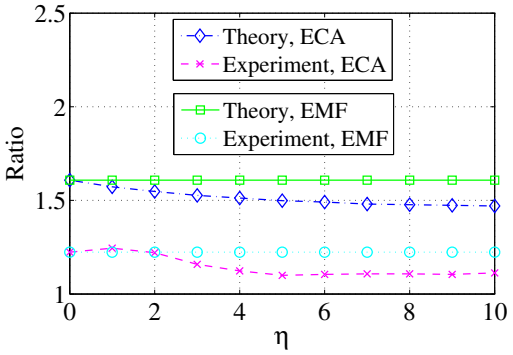


Fig. 11. ECA over OPT-B under special case, γ=0.2, μ_e=1.0, l=0.25.

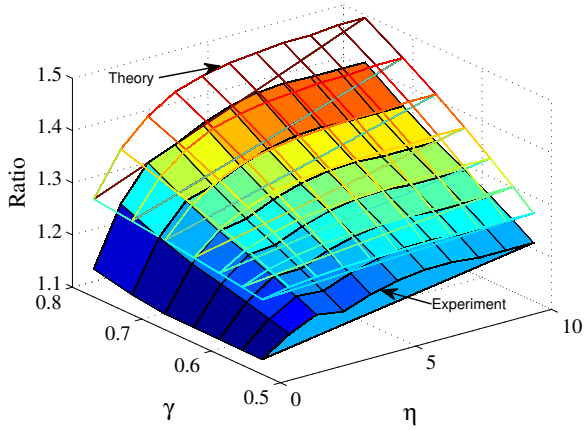


Fig. 12. Performance of ECA over OPT-C under special case, μ_e=1.0, l=0.5.

7 Conclusion

In this paper, we studied traffic flow provisioning problem by formulating it as a minimum weighted flow provisioning problem with objective of minimizing the total cost of TCAM occupation and remote packet processing. This problem is proved to be NP-hard, and an efficient heuristic algorithm is proposed to solve this problem when network traffic is given. We further propose two online algorithms to approximate the optimal solution when network traffic information is unknown in advance. Finally, extensive simulations were conducted to validate the performance of theoretical analysis of the proposed algorithms, using the real traffic traces.

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," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[2] M. Casado, M. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, and S. Shenker, "Rethinking enterprise network control," *IEEE/ACM Transactions on Networking*, vol. 17, no. 4, pp. 1270–1283, Aug 2009.

[3] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the one big switch abstraction in software-defined networks," *Proc. ACM CoNEXT*, 2013.

[4] "Openflow switch specification," https://www.opennetworking.org/sdn-resources/onf-specifications/openflow.

[5] T. Benson, A. Anand, A. Akella, and M. Zhang, "Microte: Fine grained traffic engineering for data centers," in *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies (CONEXT)*, 2011, pp. 1–12.

[6] S. Agarwal, M. Kodialam, and T. Lakshman, "Traffic engineering in software defined networks," in *IEEE International Conference on Computer Communications (INFOCOM)*, April 2013, pp. 2211–2219.

[7] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2013, pp. 545–549.

[8] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[9] U. C. Kozat, G. Liang, and K. Kokten, "On diagnosis of forwarding plane via static forwarding rules in software defined networks," in *2014 Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*. IEEE, 2014.

[10] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (cam) circuits and architectures: A tutorial and survey," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, 2006.

[11] Y. Sun and M. S. Kim, "Tree-based minimization of tcam entries for packet classification," in *7th IEEE Consumer Communications and Networking Conference(CCNC),*. IEEE, 2010, pp. 1–5.

[12] N. Katta, J. Rexford, and D. Walker, "Infinite cacheflow in software-defined networks," Tech. Rep. TR-966-13, Department of Computer Science, Princeton University, Tech. Rep., 2013.

[13] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks." in *CNSM*, 2013, pp. 18–25.

[14] S. Yang, J. Kurose, and B. N. Levine, "Disambiguation of residential wired and wireless access in a forensic setting," in *IEEE International Conference on Computer Communications (INFOCOM)*. IEEE, 2013, pp. 360–364.

[15] E. Spitznagel, D. Taylor, and J. Turner, "Packet classification using extended tcams," in *Proceedings of 11th IEEE International Conference on Network Protocols (ICNP)*. IEEE, 2003, pp. 120–131.

[16] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 351–362, 2010.

[17] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Openflow: Meeting carrier-grade recovery requirements," *Computer Communications*, vol. 36, no. 6, pp. 656–665, 2013.

[18] F. Dürr, "Towards cloud-assisted software-defined networking," Technical Report 2012/04, Institute of Parallel and Distributed Systems, Universität Stuttgart, Tech. Rep., 2012.

[19] X. N. Nguyen, D. Saucez, C. Barakat, T. Thierry *et al.*, "Optimizing rules placement in openflow networks: trading routing for better efficiency," in *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN 2014)*, 2014.

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

[21] R. Bar-Yehuda and S. Even, "A linear-time approximation algorithm for the weighted vertex cover problem," *Journal of Algorithms*, vol. 2, no. 2, pp. 198–203, 1981.

[22] B. Korte and J. Vygen, *Combinatorial optimization: Theory and Algorithms (5 ed.)*. Springer, 2012.

[23] G. Optimization, "Gurobi optimizer reference manual," 2013.