FlowStat: Adaptive Flow-Rule Placement for Per-Flow Statistics in SDN

Samaresh Bera, *Student Member, IEEE,* Sudip Misra, *Senior Member, IEEE,*
and Abbas Jamalipour, *Fellow, IEEE,*

***Abstract*—In this paper, we propose an adaptive flow-rule placement scheme, FlowStat, in software-defined network (SDN) with an aim to provide per-flow statistics to SDN controller, while enhancing overall network performance. The proposed scheme consists of three phases — forwarding path selection, flow-rule placement, and rule redistribution. In the first phase, we formulate a *max-flow-min-cost* optimization problem to determine *optimal* forwarding paths while considering multi-commodity flows with heterogeneous requirements. In the second phase, an integer linear programming (ILP) problem is formulated to decide forwarding rules for paths computed in the first phase, so that total number of exact-match is minimized. As finding optimal solution to the problems is NP-hard, we propose two greedy heuristic approaches to solve the problems in polynomial time. Finally, we propose a rule redistribution scheme on detecting rule congestion at a switch, in order to accommodate new flows in the network. Extensive experimental results show that the proposed scheme, FlowStat, is capable of providing per-flow statistics to the SDN controller, while enhancing network performance compared to existing schemes — ReWiFlow and ExactMatch. In particular, FlowStat is capable of reducing end-to-end delay and QoS violation by 46% and 75% (approx.), respectively, compared to the ReWiFlow and ExactMatch schemes, while providing 85% accurate per-flow statistics to the SDN controller.**

***Index Terms*—Software-defined networks, Per-flow statistics, rule placement, optimization**

I. Introduction

The growing concerns about digitization of everything necessitate the current *best-effort* Internet technology to be modified, in order to support new services and applications [1], [2]. Further, over-provisioning of bandwidth in current Internet increases CAPEX and OPEX to service providers, while imposing different challenges in supporting new applications and services. This is due to the *vendor-specific* architecture of forwarding devices (i.e., switches and routers). Software-defined network (SDN) architecture is a viable approach to address the limitations of the current Internet while decoupling the control-plane from forwarding devices [3]–[6]. In the control-plane, a logically centralized controller controls the forwarding devices by deciding control-logics and placing them at the devices in the form of flow-rules [2], [7]. Consequently, the SDN architecture provides a logically centralized view of the network which yields better utilization of network resources and improved network management. Further, network function virtualization (NFV) is also introduced for placing network functions at the network components in real-time [8], [9].

A. Motivation

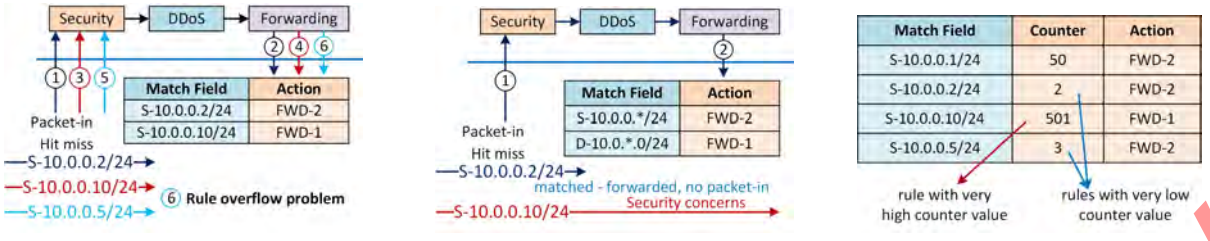
The flow-rules defined by the SDN controller are installed at the switches by utilizing ternary content-addressable memory (TCAM). The TCAM available at a switch is limited due to high cost and energy consumption. As a result, due to limited TCAM, number of flow-rules that can be inserted at a switch is also limited. To place flow-rules at the switches, researchers proposed three different strategies — exact-match, wildcard, and hybrid. In exact-match, each flow[1] is associated with an individual rule, and thereby, increases network visibility. However, such an approach leads to frequent rule replacement, thereby increasing controller overhead and network delay. On the other hand, in wildcard-based strategy, multiple flows are associated with a few flow-rules. Although the wildcard approach reduces the requirement of frequent rule replacement, it decreases the network visibility, which, in turn, leads to incorrect forwarding decisions and QoS violations. In hybrid approach, combination of exact-match and wildcard is considered. Figure 1 presents the issues present in SDN from the aspects of flow-rule placement. Figure 1(a) presents an example of rule overflow problem due to exact-match rule placement. In exact-match, to accommodate a new flow, existing rules are deleted irrespective of whether they are active or not. Consequently, more flow-rule installation requests are generated in the form of Packet-In[2] messages, which, in turn, increases controller overhead. In contrast, the controller does not have exact flow information in case of wildcard rule placement, as depicted in Figure 1(b). When an incoming packet is matched with an existing rule, it is forwarded according to the desired action without notifying the SDN controller. As presented in Figure 1(b), the packet which comes later matches with the existing rule and is forwarded in the network. However, this particular packet can be malicious to the network. Thus, security concerns are required to be considered while placing wild-card rules at the switches. Moreover, a network supporting heterogeneous applications includes both *mice* and *elephant flows*, which may lead to inefficient rule-space utilization, as depicted in Figure 1(c). Motivated by these facts, we intend to propose an adaptive

S. Bera, S. Misra are with the Computer Science and Engineering Department, Indian Institute of Technology, Kharagpur, 721302, India, Email: s.bera.1989@ieee.org, smisra@cse.iitkgp.ac.in

A. Jamalipour is with the School of Electrical & Information Engineering, University of Sydney, Australia, Email: a.jamalipour@ieee.org

[1]A flow is defined as stream of packets.

[2]A Packet-In message is generated when an incoming flow does not match with the flow-rules installed at the switch. It contains *meta-data* of the packet. Interested readers may refer to [10] for details.



(a) Exact-match: rule OVERFLOW (b) Wildcard: UNSEEN flows (c) INEFFICIENT rule-space utilization

Fig. 1: Issues in TCAM-based rule placement in SDN

flow-rule placement scheme while considering the advantages of both the exact-match and wildcard approaches.

B. Contribution

We propose an adaptive flow-rule placement scheme with an aim to maximize the number of flows that can be accommodated in the network, while increasing network visibility. The proposed scheme consists of three phases — forwarding path selection, flow-rule placement, and rule redistribution. In forwarding path selection phase, we formulate an optimization problem to select *optimal* paths for routing flows from source to destination in the network. In the second phase, we formulate another optimization problem to select *optimal* number of switches in the selected paths for exact-match rules placement, so that per-flow traffic statistics are obtained at the SDN controller. Finally, in the third phase, we propose a rule redistribution scheme on detecting rule congestion at the switches. The problem is challenging due to the rule-space and network capacity constraints, and QoS-guaranteed routing of flows in the network, while increasing network visibility. Figure 2 presents an overview of the proposed scheme. In

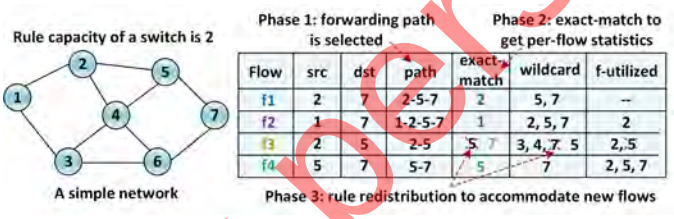


Fig. 2: Proposed solution: flows are received at the switches in the network serially. The SDN controller places rules in three phases upon receiving a new flow at a switch: (a) In the first phase, the controller determines *optimal* forwarding path to route the flow from source to destination; (b) In the second phase, the controller selects *optimal* switch in the selected path for exact-match rule placement in order to get per-flow statistics; (c) Finally, flow-rule is redistributed among the switches to accommodate new flows in the network upon detecting rule congestion at a switch.

brief, the contributions in this work are as follows.

- We formulate a *max-flow-min-cost* problem from the aspects of SDN, while considering QoS requirements of the flows in the network. We propose a greedy heuristic approach to solve the problem in polynomial time as the problem is NP-hard.
- We formulate an ILP to find *optimal* number of switches to place exact-match rules, in order to get per-flow statistics. We propose a greedy heuristic approach to solve the problem as finding *optimal* solution is NP-hard.
- We propose a flow-rule redistribution scheme on detecting rule congestion at the switches, while ensuring minimum number of redistribution of the rules. This helps to accommodate new flows in the network using the existing rule-space available at the switches.
- We evaluate the performance using Mininet network emulator and POX SDN controller in order to show the efficacy of the proposed scheme.

The rest of the paper is organized as follows. Section II presents the state-of-the-art of rule placement in SDN. In Section III, we present the detailed system architecture considered in the work with prerequisites. Section IV presents the proposed path selection, rule placement, and rule redistribution methods, while analyzing the computational complexity of the proposed scheme. Section V presents the results to show the effectiveness of the proposed scheme over existing approaches. Section VI discusses a few use-case scenarios of the proposed scheme. Finally, we conclude the work in Section VII while highlighting some future research directions.

II. Related Work

Recently, researchers proposed several schemes for flow-rule placement in SDN. We categorize the existing works from three aspects — rule placement using TCAM [11]–[15], hybrid approach [16]–[20], and flow-statistics collection [21]–[24]. In rule placement using TCAM, flow-rules are placed at the hardware switches by utilizing the available TCAM. On the other hand, combination of hardware and software switches are utilized for rule placement in hybrid approach. In flow-statistics collection, low-cost flow monitoring schemes were proposed. We discuss some of the existing approaches from the above mentioned aspects.

Giroire et al. [11] proposed an energy-aware routing scheme in SDN-enabled network. They utilized the existing rule-space available at the hardware switches for rule placement, while minimizing the link-utilization to reduce energy consumption. Similarly, Nguyen et al. [12] proposed a rule placement scheme to accommodate maximum number of flows in the network. Consequently, they focused on the destination-oriented

data delivery policy without considering the associated cost and delay in traffic forwarding. A joint optimization scheme for rule placement and traffic engineering scheme was proposed by Huang et al. [13]. The authors considered the available TCAM capacity to place flow-rules, while considering the QoS requirements of flows. Therefore, a trade-off between the rule placement and QoS-guaranteed data delivery is imposed in the network. Rifai et al. [14] proposed a rule-compression mechanism to accommodate large number of flows in the network with reduced network visibility. Recently, a mobility-aware adaptive flow-rule placement scheme was proposed, in which the flow-rules are placed at software-defined access points according to end-users' mobility in the network [15]. The authors showed that the proposed scheme is beneficial for minimizing delay and associated cost in data delivery.

Katta et al. [16] utilized the benefits of hardware and software switches for rule placement. Due to the limited TCAM, rules associated with *heavy-hitter* flows were installed at the TCAM for fast processing. On the other hand, the rules with low-counter value were placed at the software switches. Further, rules were redistributed between hardware and software switches, while considering rule-dependency problem. Similarly, Kentis et al. [19] also proposed rule placement scheme while utilizing both hardware and software switches. However, in such schemes, computational complexity is very high due to the rule dependency problem between software and hardware switches, as reported in [19]. Further, packet processing delay is increased due to the use of software switches.

Su et al. [22] proposed a low-cost flow monitoring scheme in SDN. The authors proposed an optimal polling scheme to get per-flow-rule statistics in the network, while reducing communication overhead. Kamisinski and Fung [23] proposed an efficient flow-monitoring scheme to detect anomaly switches in the network. Two types of anomalies were considered – packet dropper and swapper. In the proposed scheme, the SDN controller collects flow-statistics from the switches periodically. Further, the flow-statistics were analyzed to detect the presence of anomaly switches in the network.

Synthesis: Detailed analysis of the existing schemes reveals that there exists a research lacuna on rule placement policies for QoS-guaranteed data delivery, while providing per-flow statistics to the controller. The existing schemes either focused on the utilization of hardware and software switches for rule placement or per-flow-rule statistics collection. However, in a practical scenario, both are equally important for efficient network management in the presence of heterogeneous flows in the network. Moreover, providing per-flow statistics to the SDN controller is a crucial challenge, while considering limited TCAM available at the switches.

III. System Model

Let us consider the SDN-enabled backbone network as a directed graph $\mathcal{G}$ with rule and link capacities, i.e., $\mathcal{G} = (\mathcal{S}, E)$, where $\mathcal{S}$ is the set of switches and E is the set of edges in the graph. Each switch $i \in \mathcal{S}$ has a limited TCAM, and the switches are homogeneous in nature. Further, each link $(i,j) \in E$ has a positive capacity to carry incoming traffic. In this work, we consider that bandwidth and delay are two factors associated with a link. List of symbols used in this work is presented in Table I.

TABLE I: List of symbols

Symbol	Description
$\mathcal{S}$	Set of switches in the network
E	Set of links in the network
$C_{i,j}$	Capacity of a link (i,j), $i,j \in \mathcal{S}$
$d_{i,j}$	Delay of a link (i,j), $i,j \in \mathcal{S}$
$l_{i,j}$	Loss of a link (i,j), $i,j \in \mathcal{S}$
$\mathcal{F}$	Set of flows in the network
$F_{i,j}$	Set of flows over link (i,j)
$F_{i,j}^f$	A flow in $F_{i,j}$
f	A flow in the network
R_i^{max}	Rule capacity of a switch $i \in \mathcal{S}$
R_i^{util}	Rule capacity utilization of a switch $i \in \mathcal{S}$
$C_{i,j}^{util}$	Link capacity utilization of a link (i,j)
$\Phi_{i,j}^f$	Routing cost of flow $f \in \mathcal{F}$ over link (i,j)

A. Prerequisites

Definition 1. ***Single Commodity Flow****: A single commodity flow is defined as follows:*

$$F_{i,j} \leq C_{i,j}, \forall (i,j) \in E \tag{1}$$

$$\sum_{(i,j)\in(i,j)_{out(i)}} |F_{i,j}| = \sum_{(i,j)\in(i,j)_{in(i)}} |F_{i,j}|, \forall i,j \in \mathcal{S} \setminus \{s,t\} \tag{2}$$

and

$$\sum_{(i,j)\in(i,j)_{out(s)}} |F_{i,j}| = \sum_{(i,j)\in(i,j)_{in(t)}} |F_{i,j}| = |\mathcal{F}| \tag{3}$$

where $\mathcal{F}$ and $F_{i,j}$ denote the set of flows in the network and set of flows over the link (i,j), respectively. Further, s and t denote the source and destination of the flow, respectively. Equations (2) and (3) preserve the flow conservation properties.

Definition 2. ***Multi-Commodity Flow****: In multi-commodity flow, multiple source and destination are present in the same network with varying demands of the flow. It is defined as follows:*

$$\sum_{f \in F_{i,j}} \gamma^f F_{i,j}^f \leq C_{i,j}, \forall (i,j) \in E \tag{4}$$

and Eqn. (2), (3) for flow conservation.
Equation (4) confirms that the total demand of flows routed through a link is always within the link capacity. γ^f denotes the demand of the flow $f \in F_{i,j}$.

Definition 3. ***Unsplittable Flow****: A flow $f \in \mathcal{F}$ is called unsplittable flow if the set of links $(i,j) \in E \mid F_{i,j}^f > 0$ forms a simple cycle-free path from source $s \in \mathcal{S}$ to destination $t \in \mathcal{S}$. Mathematically,*

$$\sum F_{s,i}^f = \sum F_{i,j}^f = \sum F_{j,t}^f, \forall s,i,j,t \in \mathcal{S}$$

IV. Proposed Rule Placement Scheme

A. Forwarding Path Selection

One of the primary objectives of the proposed scheme is to accommodate maximum number of flows in the network, while minimizing associated cost. We design a cost function $\Phi_{i,j}$ to forward a flow over a link $(i,j) \in E$ while considering start-up cost for link activation σ, rule-utilization of the forwarding switch $i \in \mathcal{S}$, and link-utilization. Mathematically,

$$\Phi_{i,j} = \sigma b_{i,j} + \alpha \frac{R_i^{util}}{R_i^{max}} + \beta \frac{C_{i,j}^{util}}{C_{i,j}} \tag{5}$$

where $b_{i,j}$ is a boolean parameter to denote whether a new link is selected to forward the flow. Terms α and β are predefined constants used to capture application-specific requirements. The value of the constants are user-defined. We formulate a *max-flow-min-cost* problem to determine *optimal* forwarding paths while considering QoS requirements of flows and network constraints. Mathematically,

$$\max_{f} \min_{(i,j)} \sum_{f \in \mathcal{F}} \sum_{(i,j) \in E} F_{i,j}^f \Phi_{i,j} x_{i,j}^f, \forall (i,j) \in E \tag{6a}$$

$$\text{s.t.} \quad \sum_{f \in \mathcal{F}} \gamma^f x_{i,j}^f \leq C_{i,j} \tag{6b}$$

$$R_i^{util} \leq R_i^{max}, \forall i \in \mathcal{S} \tag{6c}$$

$$f^{req}[d, l, -\gamma] x_{i,j}^f \succeq [d_{i,j}, l_{i,j}, -C_{i,j}] \tag{6d}$$

$$\text{Eqn. } (2) \text{ and } (3) \tag{6e}$$

The objective function denotes that minimum number of links are selected while accommodating maximum number of flows in the network, in order to minimize associate cost. $\Phi_{i,j}$ denotes the cost associated with the link (i,j) to route a flow, as presented in Equation (5). Equation (6b) ensures the link capacity constraint, i.e., total required capacity of the flows that can be routed through a link (i,j) is always less than or equal to the capacity of the link. Similarly, the number of flow-rules installed at a switch $i \in \mathcal{S}$ is always less than or equal to the maximum number of rules that can be inserted at the switch due to limited TCAM, as denoted in Equation (6c). Further, application-specific requirements are considered in Equation (6d) while routing the flows in the network. The tuple $< d, l, \gamma >$ denotes the delay, loss, and bandwidth requirements of a flow, and it should be fulfilled while choosing a link (i,j) to route the flow. We use the notations $-\gamma$ and $-C_{i,j}$ to denote component-wise inequality between vectors [25].

The optimization problem in Equation (6a) consists of two integer linear programming (ILP) problems. The first ILP is about the maximization of flows that can be accommodated in the capacitated network while considering link-capacity and rule-capacity constraints. On the other hand, the second ILP is about the minimization of associated cost, while considering flow-requirements, link-capacity, and rule-capacity. Mathematically,

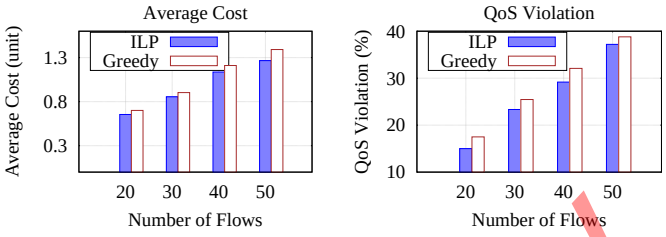


Fig. 3: Performance comparison between ILP and proposed greedy approach

P1: with fixed capacity

$$\max \sum_{f \in \mathcal{F}} \sum_{(i,j) \in E} F_{i,j}^f x_{i,j}^f \tag{7}$$

s.t. Eqn (2) and (3)
Eqn. (6b), (6c) and (6d)

P2: with fixed no. of flows

$$\min \sum_{f \in \mathcal{F}} \sum_{(i,j) \in E} \Phi_{i,j} x_{i,j}^f \tag{8}$$

s.t. Eqn. (2), (3)
Eqn. (6b), (6c) and (6d)

Solving the both the optimization problems in polynomial time is NP-hard in general, while considering QoS requirements of flows [26]. In other words, *optimal* solution to the problem cannot be found in polynomial time, while considering multi-constrained QoS requirements of flows in the network. We limit our discussion on NP-hard problem as it is well-explored in the literature [26]. To solve the problem in polynomial time, we propose a greedy heuristic approach for path selection to forward incoming flows by considering the associated constraints, while reducing overall associated cost.

Algorithm 1 presents the specific algorithm for forwarding path selection. The proposed algorithm determines the paths to forward incoming flows while ensuring QoS requirements of the latter. The Yen's K-shortest path algorithm [27] is used in Step 3 to get k number of shortest paths in the network for a given flow. The paths in the K-shortest paths are sorted in ascending order based on the associated cost. Therefore, better path is always preferred from the K-shortest paths to forward incoming flows. When no QoS path is found, the k-th path is chosen to forward the traffic, as presented in Step 7. It ensures that the algorithm will work even there is no QoS path found to forward the traffic. However, in such a situation, the performance of the proposed scheme is degraded in terms of delay, loss, and throughput. If we have a large value for k, then the algorithm would take more time to determine K-shortest paths, which, in turn, would increase the end-to-end delay. On the other hand, if we have a small value for k, we may not get QoS satisfied path from the obtained K-shortest paths. Therefore, there exists a trade-off for selecting the value of k. In the proposed scheme, we select k=3 for forwarding path selection.

Figure 3 presents a performance comparison between ILP and proposed greedy approach for forwarding path selection. It is evident that the proposed greedy scheme yields competitive performance to the ILP. Further, in case of ILP, the computation time increases exponentially with increasing number of

Algorithm 1 Forwarding path selection algorithm

Inputs: Network Graph, $\mathcal{G}$, with link and node parameters
Set of flows $\mathcal{F}$ with requirements
Values for constants $\sigma_{i,j}$, α, and β ▷ User defined
Output: Set of Paths $\mathcal{P}$ on which flows $\mathcal{F}$ can be routed

$k \leftarrow 1$
while all flows $f \in \mathcal{F}$ are not assigned paths **do**
  for P in K-SHORTEST-PATHS(s, t) **do**
  ▷ K number of shortest paths based on cost in Eqn. (5)
    if CHECK-QOS(P, f) **then** ▷ QoS satisfied
      flag = 1
      P_k = P ▷ through which f to be routed
  if flag $\neq$ 1 **then** ▷ QoS path not found
    P_k = $P - 1$
  $\mathcal{P} \leftarrow append(P_k)$
  UPDATE_CAPACITY(P_k, f) ▷ update link-capacity
  $k \leftarrow k + 1$ ▷ k^{th} flow to be routed using path P_k
function CHECK-QOS(P, f)
  for (i, j) in P **do**
    if $f_{req}[d, l, -\gamma] \geq [d_{ij}, l_{ij}, -C_{ij}]$ **then**
      return True
    else
      return False
function UPDATE_CAPACITY(P, f)
  for (i, j) in P **do**
    $C_{i,j} = C_{i,j} - \gamma^f$ ▷ capacity is reduced by γ^f

flows.

B. Rule Placement at Switches

After selecting the paths to route the flows, we need to select *optimal* number of switches, in which exact-match rules should be installed, in order to get per-flow statistics in the network. In other words, the objective is to minimize the number of exact-match rules over all paths in order to reduce rule-space utilization, and hence, accommodate more flows. Consequently, we formulate an ILP to select the switches to place exact-match rules in the network for the set of paths obtained in Section IV-A. Mathematically,

$$\min \quad \sum_{k=1}^{|\mathcal{P}|} \sum_{i \in P_k} M_k^{exact} y_i \tag{9a}$$

$$\text{s.t.} \quad \sum_{i \in P_k} M_k^{exact} y_i \geq 1, \forall k \in |\mathcal{P}| \tag{9b}$$

$$\sum_{j \in \mathcal{S}} x_{i,j} \leq \sum_{k \in |\mathcal{P}|} M_k^{exact} y_i, i \in \mathcal{S}, i \neq j \tag{9c}$$

y_i denotes a boolean variable whether a switch $i \in \mathcal{S}$ is selected to place exact-match. Equation (9b) denotes that atleast one exact-match is placed in a path. As a result, from the exact-match, the SDN controller gets the per-flow statistics in the network. Equation (9c) denotes that the number of flow-rules at a switch $i \in \mathcal{S}$ is greater than or equal to the number of active forwarding edges from the switch according to the residual graph obtained from the first phase. It is noteworthy that *default* flow-rules[3] are installed [28] at rest of the switches in a path associated to a flow. Similar to the optimization problem presented in Section IV-A, the above optimization problem is also NP-hard, due to the complexity involved in computation of all possible combinations of the candidate switches and processing those for exact-match rule placement. Consequently, we propose a greedy heuristic to select switches in the selected paths, in order to place exact-match. As the objective is to maximize flow visibility while considering rule-capacity, we consider degree, $deg(i)$, of a switch $i \in \mathcal{S}$ and its utilized rule-capacity, R_i^{util}. To consider a trade-off between $deg(i)$ and R_i^{util}, we introduce a *network visibility factor*, λ, defined in Definition 4. It is noteworthy that the value of λ will impact on the selection of switches in which exact-match is placed.

Definition 4. ***Network visibility factor****: It is the desired weight, λ, on the degree of a switch, $deg(i)$, $i \in \mathcal{S}$, to place exact-match at switch i, where $0 < \lambda < 1$ and $\lambda \in \mathbb{R}_{>0}$.*

In a practical scenario, the switch with higher degree should be prioritized to avoid congestion over a specific link, while considering rule-space utilization of the switch. Consequently, we consider an eligibility score, $T_i(P)$, of a switch for exact-match placement is calculated as follows:

$$T_i(P) = \lambda \frac{deg(i)}{\Delta(\mathcal{G})} - (1 - \lambda) \frac{R_i^{util}}{R_i^{max}}, \forall i \in \mathcal{S}, \text{ and } P \in \mathcal{P} \tag{10}$$

where $\Delta(\mathcal{G})$ denotes the maximum degree of the network. Equation (10) ensures a trade-off between the degree and rule-capacity utilization of a switch. The proposed heuristic approach is presented in Algorithm 2 for flow-rule placement.

Algorithm 2 Flow-rule placement algorithm

Inputs: Set of degrees of all switches
Set of paths with flows $< \mathcal{P}, \mathcal{F} >$
Maximum rule-capacity of the switches $\mathcal{R}_i^{max}, \forall i \in \mathcal{S}$
Set of utilized rule-capacity $\mathcal{R}^{util} = \{R_i^{util}\}, \forall i \in \mathcal{S}$
Output: Place flow-rules at switches to route incoming flows

$k \leftarrow 1$
while all paths $P_k \in \mathcal{P}$ are not assigned flow-rules **do**
  get switch $i \in P_k$ with max. score using Eqn. (10)
  get exact-match at i for flow $f_k \in \mathcal{F}$
  get *default* rules at switches $j \in P_k \setminus i$
  place the exact-match and *default* rules
  update $R_i^{util}, \forall i \in P_k$ ▷ update rule utilization
  $k \leftarrow k + 1$ ▷ Rules for k^{th} path is installed

C. Flow-rule Redistribution

In this section, we propose a two-stage rule redistribution approach to accommodate more flows in the network – path- and flow-based redistribution. In path-based redistribution, rules associated with a flow are redistributed among the switches in the existing path itself, without exploring a new

[3]Source (nw_src) and destination (nw_dst) are considered by default.

path. On the other hand, in flow-based redistribution, new paths associated with existing flows are explored. Accordingly, flow-rules are placed at the switches in the new paths until the utilized rule-capacity of all switches has reached a predefined threshold. The first-stage ensures minimum rule redistribution in the network, as only exact-match rules are required to be distributed. If the first-stage fails, the second-stage explores new paths and places flow-rules while considering rule-dependency problem. The rule redistribution algorithm runs periodically at the controller-end, and, accordingly, flow-rules are updated at the switches in the network. Algorithm 3 presents the proposed rule redistribution scheme. In step

Algorithm 3 Flow-rule redistribution algorithm

Inputs: Set of paths with flows and exact-match switches $< \mathcal{P}, \mathcal{F}, \mathcal{M} >$
Rule-congested switch $v \in \mathcal{S}$
Output: Flow-rule placement at switches to route flows in the network

1: $F \leftarrow$ set of flows routed though S
2: $P \leftarrow$ set of paths associated with F
3: flag $\leftarrow 0$ ▷ used to check rule-utilization threshold
First stage: ▷ redistribute using existing paths
4: **for** path $p \in P$ **do**
5: **for** switch $s \in p \setminus v$ **do**
6: **if** $R_s^{util} \leq R^{th}$ **and** $M_p^{exact}(v) \wedge M_p^{wild}(v)$ **and** $f_p(s) \wedge f_p(v)$ **then**
7: place M_p^{exact} in s
8: delete M_p^{exact} from v
9: **if** $R_v^{util} \leq R^{th}$ **then**
10: flag $\leftarrow 1$
Second stage: ▷ update rules for new paths
11: **if** !flag **then**
12: **for** all flow $f \in F$ **do**
13: get alternate path $p^{'}$ using Algorithm 1 (Steps 3–8)
14: get flow-rules for $p^{'}$ using Algorithm 2 (Steps 3–5)
15: **for** switch $s \in reverse(p^{'})$ **do** ▷ reverse path for consistent update
16: **if** ! OFPFF_CHECK_OVERFLAP **then**
17: place flow-rule at s

15, the rules are updated at the switches in a reverse order in the path for consistent update [29]. Therefore, some fraction of a flow is forwarded using old rule and some follows the updated rule until all the switches in the path are updated. Further, this ensures *loop-free* and connected network update. Step 16 utilizes the *rule-overlap* utility of OpenFlow protocol [30] during flow-rule installation. It is noteworthy that the SDN controller periodically checks for rule-congestion at the switches and redistributes the flow-rules using Algorithm 3. Further, the rule update mechanism is considered in the proposed greedy heuristic approach to accommodate more number of flows in the network. Figure 4 presents the proposed controller architecture.

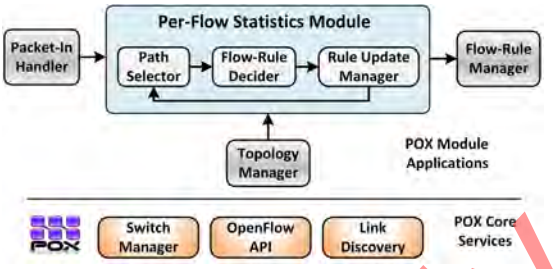
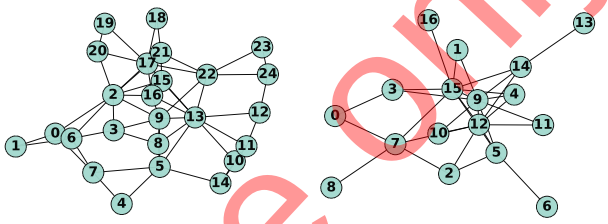


Fig. 4: Proposed SDN controller architecture



(a) AttMpls topology (b) Goodnet topology

Fig. 5: Network topologies considered in this work

D. Computational Complexity

We analyze the time complexity of the proposed scheme in three phases — path selection (Algorithm 1), rule placement (Algorithm 2), and rule redistribution (Algorithm 3). In Algorithm 1, the most computationally expensive operation is the calculation of k-shortest paths. We use Yen's K-shortest path algorithm [27], and it takes $O(K \times |\mathcal{S}|(|E|+|\mathcal{S}|log|\mathcal{S}|))$. Therefore, for $|\mathcal{F}|$ number of flows, Algorithm 1 takes $O(|\mathcal{F}| \times (K \times |\mathcal{S}|(|E|+|\mathcal{S}|log|\mathcal{S}|)))$ time. Further, Algorithm 2 involves sorting of switches for $|\mathcal{P}|$ number of paths. Therefore, it runs in $O(|\mathcal{P}| \times |\mathcal{S}|log|\mathcal{S}|)$ time. Finally, in Algorithm 3 involves two phases. The first phase runs in $O(|\mathcal{P}| \times |\mathcal{S}|)$, and the second phase runs in $O(|\mathcal{F}| \times |\mathcal{S}|)$ time. Therefore, total time complexity of the proposed scheme is $O(|\mathcal{F}| \times (K \times |\mathcal{S}|(|E| + |\mathcal{S}|log|\mathcal{S}|)))$ + $O(|\mathcal{P}| \times |\mathcal{S}|log|\mathcal{S}|)$ + $O(|\mathcal{P}| \times |\mathcal{S}|)$ + $O(|\mathcal{F}| \times |\mathcal{S}|) \approx O(|\mathcal{F}| \times (K \times |\mathcal{S}|(|E| + |\mathcal{S}|log|\mathcal{S}|)))$.

V. Performance Evaluation

We evaluate the performance of the proposed scheme, FlowStat, using POX[4] SDN controller and Mininet [31] network emulator. Further, two real-life network topologies — AttMpls and Goodnet — from the Internet topology Zoo [32] are considered in this work. The AttMpls topology is relatively dense compared to the Goodnet topology. Therefore, we choose these two topologies to consider two different aspects of network topology in a practical scenario. It is noteworthy that other network topologies may also be considered to validate the proposed scheme. However, due to high resource requirement of the emulated environment, we choose a well-balanced mixture of network topologies – AttMpls (large and dense) and Goodnet (small and sparse). Figure 5 shows the representational view of the two network topologies. Incoming network traffic with different type of services is generated

[4]https://github.com/noxrepo/

TABLE II: Simulation Parameters

Parameter	Value
Network Topology	AttMpls and Goodnet [32]
Number of switches	25 (AttMpls) and 17 (Goodnet)
Number of links	57 (AttMpls) and 31 (Goodnet)
Number of flows	100 – 300
Flow bandwidth	0.20 – 0.40 kbps
Average packet size	94 – 699 bytes [35]
Active volume	142 – 27716 bytes [35]
Mean rate	562 – 516,540 bps [35]
Active time	1 – 34 s [35]
$[\sigma, \{\alpha, \beta\}, \lambda]$	[0.005, {0.0 – 1.0}, 0.5]

using D-ITG generator [33]. The experiment is conducted in a Google Cloud[5] instance with Intel Skylake CPU and 7.5GB RAM running Linux kernel 4.4.0-103-generic. Table II presents the parameters and their values used in the experiment. The presented results are taken as average of 10 independent runs. Additionally, 95% *confidence interval* is used to show the variance of the results [34], i.e., in 95% cases, we are confident that the obtained results lie between the specified range. We also vary the values of the predefined constants α and β to show their impact on the performance of the proposed scheme.

We compare the proposed scheme, FlowStat, with existing schemes — ReWiFlow [36] and ExactMatch. In case of ReWiFlow, the flow-rules are placed at the switches based on wildcard in which source and destination of an incoming packet are considered. Rest of the fields are considered as *do-not-care*. On the other hand, in case of ExactMatch, all match-fields are considered for rule placement. In contrast to the ReWiFlow and ExactMatch, the proposed scheme, FlowStat, decides the flow-rules *intelligently* based on the Algorithms 1 and 2. Further, FlowStat redistributes the flow-rules based on real-time situations. Henceforth, we use FlowStat to represent the proposed scheme, and ReWiFlow and ExactMatch to represent the existing schemes.

A. Performance Metrics

In this Section, we discuss the performance metrics used to compare the performance of the proposed scheme with the existing schemes.

1) Accuracy in per-flow statistics: We measure the accuracy of per-flow statistics as the ratio between the number of unique flows detected by SDN controller and the total number of unique flows in the network.

2) Number of Packet-In: The number of Packet-In is measured as the number of Packet-In requests received by the controller. This is used to show the control overhead in the network. We use *default* header format for Packet-In as specified in OpenFlow [37].

3) QoS violated flows: A flow is treated as QoS violated flow when at-least one of its requirements – delay, loss or bandwidth – is not fulfilled, while forwarding the flow from source to destination. It is noteworthy that the number of QoS violated flows depends on the forwarding path selection, rule placement, and rule redistribution techniques.

[5]https://cloud.google.com/

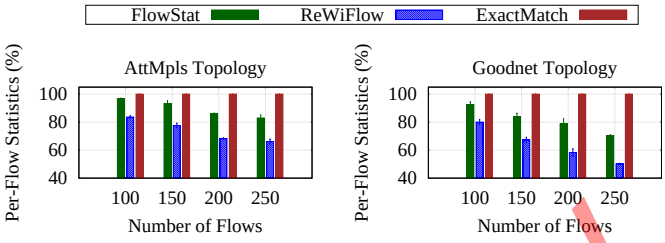


Fig. 6: Per-flow statistics with different number of flows

4) End-to-end delay: End-to-end delay is measured as the total time taken to deliver a packet from source to destination. This includes queuing, processing, transmission, and propagation delays.

5) Throughput: Throughput is calculated as the effective bandwidth usage of the active links determined in the forwarding path selection phase.

6) Packet drop: Packet drop is calculated as the ratio between the number of lost packets and total number of packets. We take percentage of the ratio to present the results.

It is noteworthy that we calculate end-to-end delay, throughput, and packet drop using the utilities available in Mininet network emulator.

B. Results and Discussion

In this section, we present the results obtained using the schemes — FlowStat (Proposed), ReWiFlow, and ExactMatch — using different performance metrics. The results are presented for AttMpls and Goodnet network topologies with different number of flows in the network.

1) Accuracy in Per-Flow Statistics: The primary objective of the proposed scheme is to obtain per-flow statistics while minimizing associated cost. As mentioned in Section IV, FlowStat places the forwarding rules at the switches in such a manner that per-flow statistics at the controller is maximized. It is noteworthy that the SDN controller analyzes the per-flow statistics using Packet-In messages only, so that additional control overhead in flow-statistics collection is avoided. Figure 6 presents the percentage of per-flow statistics accurately analyzed at the SDN controller with different schemes — FlowStat, ReWiFlow, ExactMatch. We see that the proposed scheme, FlowStat, is capable of providing approximately 85% accurate per-flow statistics. In contrast, in case of ReWiFlow, we get only 68% accuracy in per-flow statistics collection. On the other hand, ExactMatch provides 100% accuracy in per-flow statistics collection. This is due to the fact that the SDN controller places exact-match rules on receiving a new flow. Therefore, on receiving a new flow at a switch, the latter generates Packet-In to the controller, which leads to better accuracy in per-flow statistics. However, ExactMatch provides degraded network performance which is discussed in subsequent sections. It is also noteworthy that FlowStat provides improved accuracy using AttMpls topology compared to that of using Goodnet topology. Due to the relatively sparse nature of the Goodnet topology, number of outgoing ports at a switch is less compared to that of the AttMpls topology.

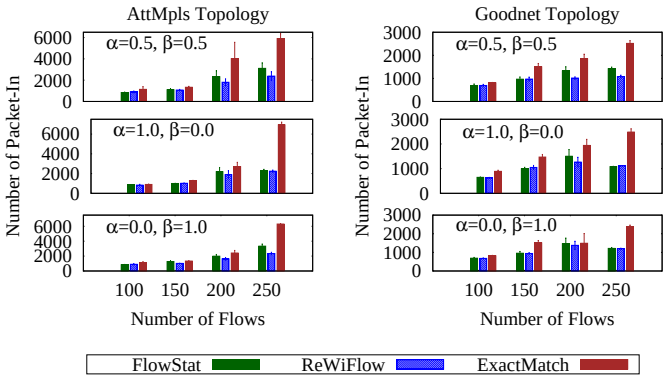


Fig. 7: Number of Packet-In with different number of flows

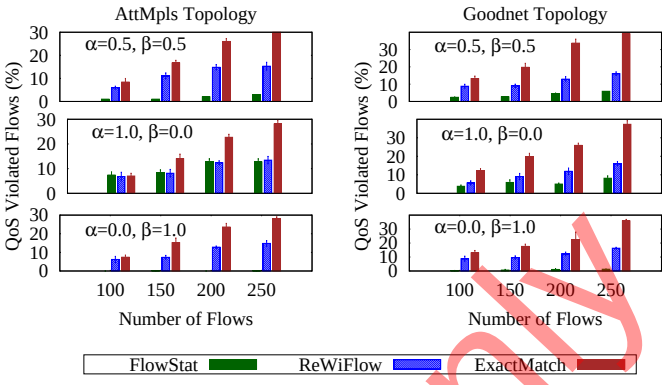


Fig. 8: QoS violation with different number of flows

Due to this reason, some of the new flows that are received at a switch match with already placed wildcard rules, which, in turn, minimizes the number of Packet-In at the controller. However, using both the topologies, FlowStat yields improved accuracy compared to ReWiFlow. Further, it can be seen that accuracy in per-flow statistics decreases with an increase in the number of flows in the network. This is due to the fact that probability of *flow-table hit* increases with increased number of flows in the network.

2) Number of Packet-in Message: As mentioned in Section IV, we use Packet-In for collecting per-flow statistics. Therefore, we also present the control overhead in per-flow statistics, as depicted in Figure 7. We see that FlowStat is capable of reducing the number of Packet-In by 40% compared to ExactMatch, while incurring 16% more Packet-In compared to ReWiFlow due to the wildcard-based rule placement strategy. Although FlowStat leads to more Packet-In compared to ReWiFlow, the former achieves improved accuracy in per-flow statistics as shown in Figure 6. Consequently, from Figures 6 and 7, it is evident that the proposed scheme, FlowStat, yields improved performance compared to the existing schemes — ReWiFlow and ExactMatch. Further, it is noteworthy that number of Packet-In remains the same for different values of α and β as the former depends on the flow-rules installed at the switches, as presented in Algorithm 2.

In the subsequent sections, we present the efficacy of the proposed scheme in terms of network performance — QoS violated flows, end-to-end delay, throughput, and packet drop.

3) QoS Violated Flows: In addition to per-flow statistics, one of the other objectives of the proposed scheme is to maintain QoS requirements of flows in the network. We propose a greedy-heuristic approach to route the traffic, while minimizing associated cost (refer to Section IV). As a result, there is no guarantee that all the traffic can be routed through the network, while preserving the QoS requirements. Consequently, we have some QoS-violated flows that are routed through the network. Figure 8 shows the percentage of QoS violation with different number of flows using AttMpls and Goodnet network topologies. We see that the proposed scheme is capable of fulfilling QoS requirements using both the network topologies. In particular, the proposed scheme, FlowStat, is capable of reducing the QoS violation by 85% and 91% (with AttMpls) and 67% and 87% (with Goodnet) compared to ReWiFlow and ExactMatch, respectively. In case of ReWiFlow and ExactMatch, the SDN controller computes the forwarding path using *open shortest path first* (OSPF) principle. Consequently, the controller considers a forwarding path as valid if network capacity is satisfied, which, in turn, leads to QoS violation. On the other hand, FlowStat computes a forwarding path while considering network capacity and QoS requirements, which leads to less QoS violation. The QoS violation is more using Goodnet network topology compared to that of using AttMpls as possibility of having alternate end-to-end path is less due to sparse nature of the former. However, it is always better than the existing schemes — ReWiFlow and ExactMatch. Further, it is observed that FlowStat yields equivalent performance in terms of QoS violation when link utilization is not considered while choosing a path. This leads to more link congestion, which, in turn, increases the QoS violation for the proposed scheme, FlowStat.

4) End-to-End Delay: Figure 9 presents the end-to-end delay in packet delivery with different number of flows using AttMpls and Goodnet network topologies, respectively. We see that the proposed scheme, FlowStat, incurs less delay in packet forwarding compared to the existing schemes — ReWiFlow and ExactMatch. In particular, FlowStat is capable of reducing end-to-end delay approximately by 48% and 44% using AttMpls and Goodnet topologies, respectively, compared to ReWiFlow and ExactMatch. In FlowStat, the SDN controller takes decision on forwarding path selection while considering delay requirements of incoming flows. Further, FlowStat *intelligently* places the flow-rules at the switches, so that less number of Packet-In is generated. This leads to less flow-setup delay compared to ExactMatch. On the other hand, we observe that a particular set of switches are congested in case of ReWiFlow due to OSPF-based path selection mechanism, which, in turn, leads to increased end-to-end delay. Another interesting fact is that ExactMatch and ReWiFlow incur almost the same end-to-end delay. In ExactMatch, incoming flows are forwarded through multiple outgoing ports according to flow-rule installed at the switches, which leads to less congestion on a specific outgoing port. In contrast, a specific outgoing port is congested due to wildcard-based rule placement. Consequently, we get almost similar performance using ReWiFlow

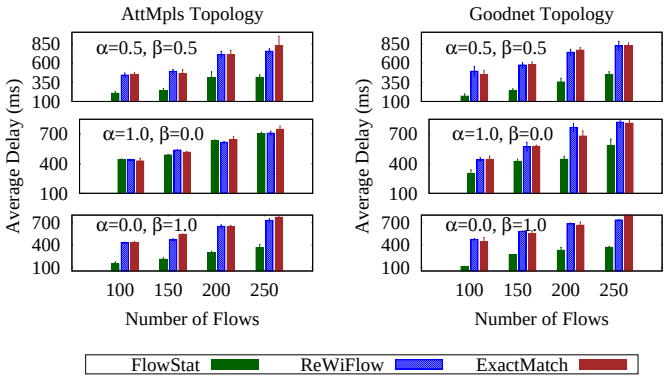


Fig. 9: End-to-end delay with different number of flows

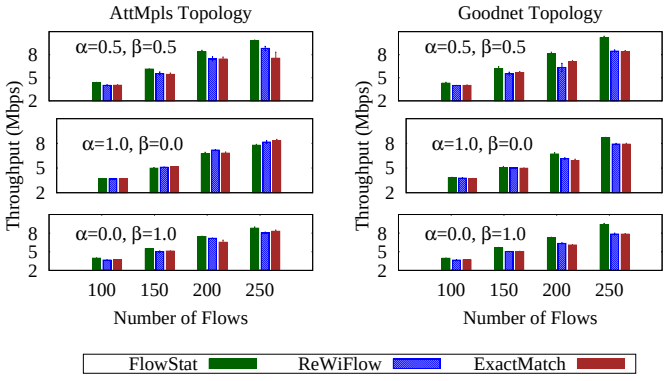


Fig. 10: Network throughput with different number of flows

and ExactMatch although the former generates less number of Packet-In (refer to Figure 7). However, the proposed scheme, FlowStat, always yields improved performance compared to the existing schemes. Similar to QoS violation, end-to-end delay also increases when link utilization is not considered during forwarding path selection.

5) Throughput: We also computed the bandwidth utilization with different number of flows using AttMpls and Goodnet topologies, as depicted in Figure 10. We see that the proposed scheme, FlowStat, achieves 18% and 14% (with AttMpls), 18% and 28% (with Goodnet) increased bandwidth utilization compared to ReWiFlow and ExactMatch, respectively. Further, we observe that the bandwidth utilization increases with an increase in the number of flows in the network as more number of flows are routed in the network within the same time. However, some of the flows are dropped with an increase in the number of flows (refer to Section V-B6), which may lead to inefficient bandwidth utilization.

6) Packet Drop: Finally, we computed the percentage of packet-drop with different number of flows using AttMpls and Goodnet topologies. Figure 11 depicts the percentage of packet-drop in the network with different number of flows. It is evident that the proposed scheme is capable of reducing the packet-drop in the network compared to the existing schemes — ReWiFlow and ExactMatch. In case of ReWiFlow and ExactMatch, more number of packets are dropped due to link

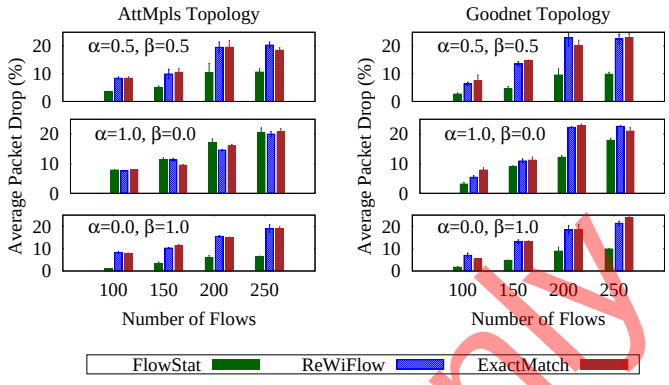


Fig. 11: Packet drop with different number of flows

congestion and increased control overhead, respectively, as discussed in Section V-B4. Further, we see that the percentage of packet-drop increases with an increase in the number of flows using FlowStat. However, it is always better than the existing schemes. Further, we observe that the packet drop increases using the proposed scheme when link congestion is not considered. This leads to more packet congestion, and eventually, more number of packets gets dropped. It is noteworthy that we do not consider application layer retransmissions of the dropped packets in this work.

In summary, it is evident that the proposed scheme is capable of enhancing the network performance in terms of end-to-end delay, throughput, packet loss, and QoS violation compared to the existing schemes, while providing improved per-flow statistics with less controller overhead. Further, it is also observed that the performance of the proposed scheme is degraded when rule-space and link utilization are not considered during path selection phase. However, it is always better than the existing schemes.

VI. Practical Applications

In this Section, we discuss two use-case scenarios in which the proposed scheme can be beneficial to meet application-specific QoS requirements.

- *QoS-guaranteed health data delivery*: In a health-care system, major data is delay- and loss-sensitive. Therefore, the data generated from different physio-logical sensors need to delivered within the specified time-bound, while incurring minimum loss. The proposed scheme is capable of achieving such requirements by employing adequate data forwarding and rule placement mechanism. Further, it is also evident from the results that the proposed scheme is capable of minimizing end-to-end delay, while minimizing packet-drop as well.
- *Energy management in smart grid system*: In smart grid, customers send their energy consumption data to the service provider in real-time through smart meter data management system. According to the received energy consumption information, the service provider takes adequate decision for reliable and cost-efficient energy management. Such real-time energy management system requires efficient data delivery with minimum delay. On the other hand, few other applications, such as billing and making business policy, require

guaranteed data delivery with minimum loss. The proposed scheme is capable of addressing such issues present in a smart grid system.

VII. Conclusion

In this paper, we proposed an adaptive flow-rule placement scheme to collect per-flow statistics in the network, while considering associated overhead and network capacity constraints. The proposed scheme consists of three phases — forwarding path selection, rule placement, and rule redistribution. In the first phase, we formulated a max-flow-min-cost optimization problem for finding *optimal* routing paths, and proposed a greedy heuristic approach to solve the problem. In the second phase, we formulated an ILP to decide optimal flow-rules, so that number of exact-match rules is minimized in the network. Finally, we proposed a rule redistribution algorithm to accommodate more number of flows in the network. Extensive experimental results were presented to show the efficacy of the proposed scheme.

In this work, we observed that some of the new incoming flows match with already placed flow-rules, which, in turn, do not generate Packet-In to the SDN controller. This leads to inaccurate per-flow statistics. Therefore, we plan to address this limitation as a future extension of this work. Further, we also plan to validate the proposed scheme in a very large and complex network topology as a future extension of the work.

References

[1] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[2] A. Ksentini, M. Bagaa, and T. Taleb, "On Using SDN in 5G: The Controller Placement Problem," in *Proc. of the IEEE GLOBECOM*, Dec. 2016.

[3] L. G. Roberts, "A radical new router: the Internet is brokenlets fix it," *IEEE Spectrum*, July 2009.

[4] D. L. C. Dutra, M. Bagaa, T. Taleb, and K. Samdanis, "Ensuring End-to-End QoS Based on Multi-Paths Routing Using SDN Technology," in *Proc. of the IEEE GLOBECOM*, 2017, pp. 1–6.

[5] R. A. Addad, D. Dutra, M. Bagaa, T. Taleb, H. Flinck, and M. Namane, "Benchmarking the ONOS Intent interfaces to ease 5G service management," in *Proc. of the IEEE GLOBECOM*, Dec. 2018, pp. 1–6.

[6] R. A. Addad, T. Taleb, M. Bagaa, D. Dutra, and H. Flinck, "Towards Modeling Cross-Domain Network Slices for 5G," in *Proc. of the IEEE GLOBECOM*, Dec. 2018.

[7] A. Ksentini, M. Bagaa, T. Taleb, and I. Balasingham, "On using bargaining game for Optimal Placement of SDN controllers," in *Proc. of the IEEE ICC*, May 2016.

[8] M. Bagaa, T. Taleb, and A. Ksentini, "Service-Aware Network Function Placement for Efficient Traffic Handling in Carrier Cloud," in *Proc. of the IEEE WCNC*, Apr. 2014.

[9] A. Laghrissi, T. Taleb, M. Bagaa, and H. Flinck, "Towards Edge Slicing: VNF Placement Algorithms for a Dynamic & Realistic Edge Cloud Environment," in *Proc. of the IEEE GLOBECOM*, Dec. 2017, pp. 1–6.

[10] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. R. andScott Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, Apr. 2008, pp. 69–74.

[11] F. Giroire, J. Moulierac, and T. K. Phan, "Optimizing rule placement in software-defined networks for energy-aware routing," in *Proc. of the IEEE GLOBECOM*, TX, USA, 2014.

[12] X.-N. Nguyen, D. Saucez, and C. B. andThierry Turletti, "Optimizing rules placement in OpenFlow networks: trading routing for better efficiency," in *Proc. of the ACM HotSDN*, Illinois, USA, 2014, pp. 127–132.

[13] H. Huang, S. Guo, P. Li, B. Ye, and I. Stojmenovic, "Joint Optimization of Rule Placement and Traffic Engineering for QoS Provisioning in Software Defined Network," *IEEE Transactions on Computers*, vol. 64, no. 12, pp. 3488–3499, 2015.

[14] M. Rifai, N. Huin, C. Caillouet, F. Giroire, J. Moulierac, D. L. Pacheco, and G. Urvoy-Keller, "Minnie: An SDN world with few compressed forwarding rules," *Computer Networks (Elsevier)*, vol. 121, pp. 185–207, Jul. 2017.

[15] S. Bera, S. Misra, and M. S. Obaidat, "Mobi-Flow: Mobility-Aware Adaptive Flow-Rule Placement in Software-Defined Access Network," *IEEE Transactions on Mobile Computing*, 2018, DOI: 10.1109/TMC.2018.2868932.

[16] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "CacheFlow: Dependency-Aware Rule-Caching for Software-Defined Networks," in *Proc. of the ACM SOSR*, CA, USA, 2016.

[17] J.-F. Huang, G.-Y. Chang, C.-F. Wang, and C.-H. Lin, "Heterogeneous Flow Table Distribution in Software-Defined Networks," *IEEE Trans. on Emerging Topics in Computing*, vol. 4, no. 2, pp. 252–261, 2016.

[18] X. Li and W. Xie, "CRAFT: A Cache Reduction Architecture for Flow Tables in Software-Defined Networks," in *Proc. of the IEEE Symposium on Computers and Communications*, Heraklion, Greece, 2017.

[19] A. M. Kentis, A. Pilimon, J. Soler, M. S. Berger, and S. R. Ruepp, "A Novel Algorithm for Flow-Rule Placement in SDN Switches," in *Proc. of the IEEE NetSoft*, Montreal, Canada, 2018.

[20] 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*, London, UK, 2015.

[21] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: scaling flow management for high-performance networks," in *Proc. of the ACM SIGCOMM*, Ontario, Canada, 2011.

[22] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "FlowCover: Low-cost flow monitoring scheme in software defined networks," in *Proc. of the IEEE GLOBECOM*, TX, USA, 2014.

[23] A. Kamisinski and C. Fung, "FlowMon: Detecting Malicious Switches in Software-Defined Networks," in *Proc. of the SafeConfig Workshop*, Colorado, USA, 2015.

[24] H. Xu, Z. Yu, C. Qian, X.-Y. Li, and Z. Liu, "Minimizing flow statistics collection cost of SDN using wildcard requests," in *Proc. of the IEEE INFOCOM*, GA, USA, 2017.

[25] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[26] Z. Wang and J. Crowcroft, "Quality-of-service routing for supporting multimedia applications," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, pp. 1228–1234, 1996.

[27] J. Y. Yen, "Finding the K Shortest Loopless Paths in a Network," *Management Science*, vol. 17, no. 11, pp. 712–716, 1971.

[28] *ONOS Documentation*, Accessed on Oct 03, 2018. [Online]. Available: https://wiki.onosproject.org/display/ONOS/ONOS+Documentation

[29] N. P. Katta, J. Rexford, and D. Walker, "Incremental consistent updates," in *Proc. of the ACM SIGCOMM workshop on HotSDN*, Hong Kong, China, 2017, pp. 49–54.

[30] "OpenFlow Switch Specification," Open Networking Foundation, Tech. Rep., 2009, version 1.0.0.

[31] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-defined Networks," in *Proc. ACM SIGCOMM Workshop Hot Topics in Networks*, 2010, p. 19:119:6.

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

[33] A. Botta, A. Dainotti, and A. Pescape, "A tool for the generation of realistic network workload for emerging networking scenarios," *Computer Networks (Elsevier)*, vol. 56, no. 15, pp. 3531–3547, 2012.

[34] A. Hackshaw, *A Concise Guide to Clinical Trials*. Oxford, UK: BMJ, 2009, ch. Statistical formulae for calculating some 95% confidence intervals.

[35] A. Sivanathan, D. Sherratt, H. H. Gharakheili, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Characterizing and Classifying IoT Traffic in Smart Cities and Campuses," in *Proc. of the IEEE INFOCOM Workshop*, 2017, pp. 559–564.

[36] S. Shirali-Shahreza and Y. Ganjali, "ReWiFlow: Restricted Wildcard OpenFlow Rules," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 29–35, 2015.

[37] *OpenFlow Switch Specification, Version 1.3.3*, Open Networking Foundation, Sept. 2013.