DynamiTE: Dynamic Traffic Engineering in Software-Defined Networks

Samaresh Bera, Sudip Misra, and Niloy Saha
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur, 721302, India
Email: s.bera.1989@ieee.org, smisra@sit.iitkgp.ernet.in, niloysaha@iitkgp.ac.in

***Abstract*—Control overhead is an important issue in software-defined network (SDN). Specifically, PACKET-IN messages are generated and sent to SDN controller on receiving a new flow at a switch. This leads to the worst situation when the ternary content-addressable memory (TCAM) available at the switch is fully utilized. In this paper, we propose a dynamic traffic engineering scheme, DynamiTE, to minimize the control overhead at the SDN controller, while minimizing the number of PACKET-IN messages. We propose a greedy heuristic approach to determine the *optimal* number of switches required to have higher TCAM, termed as *candidate switches*, compared to the other switches in the network. On receiving a new flow, a fully occupied switch directly forwards the flow to a candidate switch without generating PACKET-IN, which, in turn, helps to minimize number of control messages in the network. Further, a packet-tagging method is applied to notify the SDN controller about the congestion occurred at the fully occupied switch. Simulation results show that the proposed approach is capable of reducing the number of PACKET-IN and congestion in the network by placing optimal number of candidate switches in the network. Particularly, the number of PACKET-IN is reduced by 10%, compared to the OpenFlow-based forwarding schemes (OFS), and the number of packets experiencing congestion in the network is reduced by 38%, compared to the randomized forwarding scheme (RFS).**

***Index Terms*—Traffic engineering, Software-defined networks, Heuristic optimization, Packet-tagging**

I. Introduction

The inherent features of software-defined networking (SDN) enable real-time programmability of networking devices, while decoupling the *control-* and *data-* plane from the traditional forwarding devices. Different control logics are decided at the control plane, while the data plane simply forwards the traffic based on the logic decided at the former. Thus, the complexity involved in network management in the SDN-enabled network is reduced significantly compared to that of the traditional networks. Dynamic traffic engineering is one of the important aspects in SDN [1], [2]. The SDN-enabled switches forward the traffic based on the policies decided by the SDN controller. Recently, Agarwal et al. [3] proposed an SDN-based traffic engineering scheme, and showed that the SDN-based approaches are capable of forwarding the traffic in a more efficient manner compared to the non-SDN networks. OpenFlow [4] and flow-table rules are the key concepts to have adequate traffic engineering in SDN. To implement the flow-table rules, fast processing memory available at the forwarding devices (such as TCAM[1]) are utilized. However, such fast processing memory available at SDN switches is limited and power hungry. Therefore, limited number of rules can be inserted at a time at the switches.

On receiving a new traffic, the switch sends a PACKET-IN message to the controller. Consequently, the controller implements the flow-rules at the switches based on the available rule capacity, while considering other parameters as well. If rule-capacity is fully utilized at the switch, an existing flow-rule is deleted, and the new one is inserted without considering the status (i.e., active or passive) of the existing flow-rule. As a result, once the switch receives a packet associated with the deleted flow-rule again, it sends the PACKET-IN to the controller, and the same procedure is followed. Consequently, control message overhead increases, as the flow-rules are deleted and inserted in a round-robin manner. Recently, Qiao et al. [5] proposed a scheme in which a switch randomly forwards the traffic to one of its outgoing ports, instead of sending the PACKET-IN to the controller, while its rule capacity is fully utilized. Therefore, control overhead between switches and controller is reduced to a certain extent. Figure 1 presents such a scenario, where the fully occupied switch has three different options to forward the traffic. However, as depicted in Figure 2, the scheme proposed by Qiao et al. [5] has the following limitations: a) as the outgoing port for an incoming packet is chosen randomly, the packet may be forwarded through the longest path, instead of the shortest one; and b) the randomly chosen switch may also be fully occupied. In such a case, the packet is further forwarded to

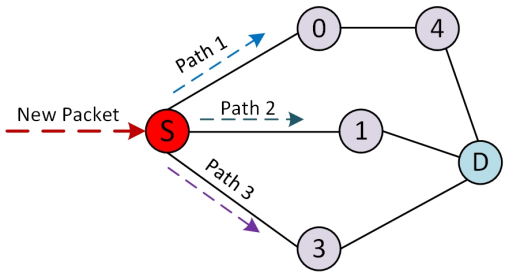

Fig. 1: An example of the scheme proposed by Qiao et al. [5]

[1]Ternary content-addressable memory (TCAM) is a fast processing memory present at SDN-enabled switches, which helps to search all available rules against an incoming traffic in a single clock cycle.

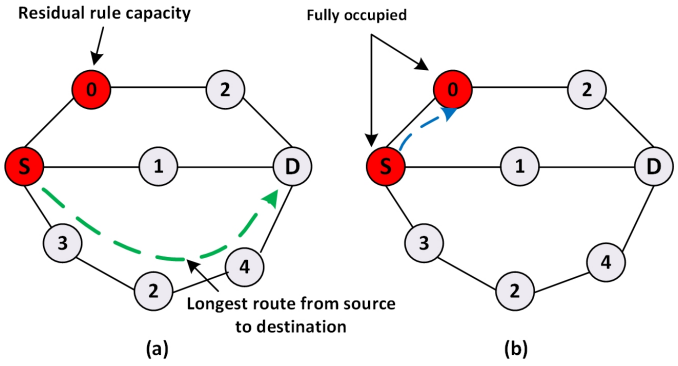

Fig. 2: Motivating scenario: Forwarding traffic to a randomly selected port may end up with — a) longest path from source to destination; and b) another fully occupied switch

a switch, which is again randomly chosen. Eventually, the packet reaches to the desired destination. In such a scenario, the controller may also be unaware of such traffic flow in the network as PACKET-IN is not sent to the former.

To deal with such a problem, we propose a dynamic traffic engineering scheme in an SDN-enabled network with an aim to forward the traffic in an adequate manner, while addressing the above mentioned issues. An optimization problem is formulated in the form of integer linear programming (ILP) to determine the minimum number of *candidate switches* required to forward the traffic for all possible affected[2] switches in the network. We propose a greedy heuristic approach to solve the optimization problem in polynomial time, as finding an optimal solution to the problem is NP-hard. Further, a packet-tagging approach is applied to notify the controller about the congestion occurred at a fully occupied switch. Simulation results show that the proposed scheme is capable of reducing number of PACKET-IN and congestion in the network, while placing optimal number of candidate switches.

The rest of the paper is organized as follows. Section II presents the state-of-the-art in dynamic traffic engineering in SDN. Detailed system model and problem description are presented in Section III. Further, the proposed approach for dynamic traffic engineering in SDN-enabled networks is presented. Section IV presents the simulation results to show the effectiveness of the proposed scheme. Finally, Section V concludes the paper while presenting some future research directions.

II. Related Work

In this Section, we present the state-of-the-art for traffic engineering in SDN. Several schemes exist in the literature for traffic engineering [3], [5]–[12]. For example, the benefits of incorporating software-defined networking concepts in traffic engineering is studied in [3]. The authors showed that the SDN-based approach is capable of forwarding network traffic in an optimal manner, while leveraging the global view of the network. Additionally, the authors formulated an optimization problem for controller placement to place flow-rules at the SDN switches deployed in the network. Caria et al. [6] analyzed the performance of network migration for traffic engineering in SDN for a given network topology. The authors proposed an algorithm to select optimal number of switches, which are required to be substituted by SDN switches, so that the need for network capacity upgradation is minimized. The proposed scheme consists of two phases — a) candidate paths selection, and b) optimization of the candidate paths. In the candidate path selection phase, all possible alternative paths are determined for a given network topology. In the second phase, the optimized paths are selected from all possible alternative paths. Consequently, the authors showed that optimal traffic engineering can be obtained while substituting a subset of the general switches by SDN switches.

Segment routing is another important aspect in traffic engineering by simplying the forwarding mechanisms. More particularly, a source node is able to specify a unicast forwarding path using the segment routing rather than specifying a shortest path, through which the packet will traverse. It is noteworthy that the segment routing was designed for SDN, while providing simplicity and better utilization of network resources in packet forwarding. Authors in [7]–[9] proposed a segment routing-based traffic engineering scheme in SDN-enabled networks, in which an architecture for segment routing is also presented. In segment routing, the SDN switches forward a packet to its next-hop switch without sending PACKET-IN to the controller. Thus, frequent rule placement at the switches can be avoided. Such an approach is useful while rule capacity of an SDN switch is nearly/completely utilized. Concurrently, Qiao et al. [5] proposed a waypoint routing scheme, while the rule capacity of the switch is fully utilized by active flows. In such a scenario, the affected switch forwards the traffic to a randomly selected neighbor without asking the controller. Consequently, the flow-rules at the switch for active flows are not replaced by the new traffic. Thus, message overhead for rule placement is avoided, while rule capacity is fully utilized.

However, *detailed analysis* of the existing approaches on traffic engineering reveals that there is a need to propose a dynamic traffic engineering scheme to deal with the issues mentioned in Section I. Therefore, we propose a dynamic traffic engineering scheme for optimally forwarding network traffic in SDN.

III. Dynamic Traffic Engineering

We consider a general SDN architecture consisting of SDN-enabled switches/routers, controller, heterogeneous constrained networks/devices, and end-users. Typically, the heterogeneous constrained networks/devices are connected to the backbone network through IoT gateways. Figure 3 shows a schematic architecture of SDN consisting of all such devices. In this work, our primary focus is traffic engineering at the

[2]Henceforth, we will use the term 'affected' for the switches, whose TCAM is fully utilized.

backbone networks, while considering incoming heterogeneous traffic in the network. The SDN switches simply forward

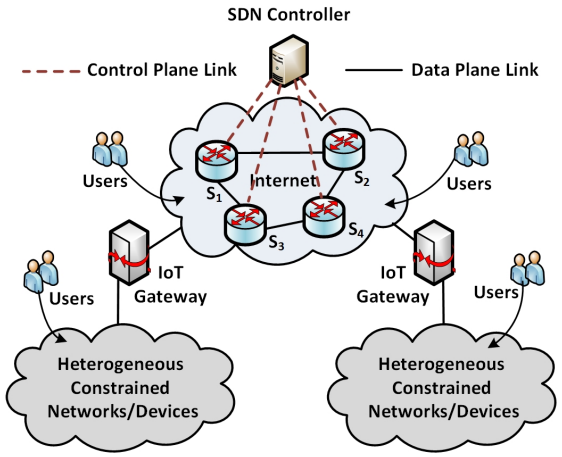

Fig. 3: A schematic architecture of software-defined network

an incoming traffic to another switch, based on the flow-rule decided by the SDN controller. If a flow-rule associated with a traffic does not exist at the switch, the switch sends a PACKET-IN to the SDN controller. Typically, the PACKET-IN message contains *Header* of the message with fields *length*, *flow-table ID*, and *data*. After receiving the PACKET-IN, the controller places adequate flow-rules at the switch, and the traffic is forwarded based on the decided policies.

Candidate Switch: We define an SDN switch, S, as candidate switch (CS), C, if it satisfies the following properties:

- Total TCAM available at the switch is higher than the TCAM available at other switches in the network. We consider a predefined value, i.e., $S_{tcam} > \Gamma$, where S_{tcam} and Γ denote the TCAM memory size and threshold value, respectively.
- The switch has more than one neighbor in its one-hop away, i.e., $|S_{neigh}| > 1$, where $|S_{neigh}|$ defines the number of neighbor switches in one-hop away.

Assumption 1. *We assume that the network is fully SDN-enabled, i.e., all the switches are SDN switches. For simplicity, we consider a single SDN controller, which controls all the switches in the network. However, multiple controllers can be placed to control the switches by forming network clusters.*

Assumption 2. *All the SDN switches support OpenFlow protocol [4] for flow-rule placement. Therefore, we utilize the benefits of the OpenFlow protocol for the proposed traffic engineering and packet-tagging approaches.*

A. Problem Statement

Let there be an SDN-enabled backbone network consisting of a set of SDN switches, which is represented as $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$, $n \in Z^+$. Our objective is to find out the subset of switches from $\mathcal{S}$ need to be configured as *candidate switches*, so that the traffic from an affected switch can be forwarded to the candidate switch without generating the PACKET-IN at the former. The set of candidate switches is represented as $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$, where $k \in Z^+$ and $\mathcal{C} \subseteq \mathcal{S}$. Mathematically,

$$\text{Minimize} \quad \sum_i C_i$$

subject to

$$\sum_{i \in \mathcal{C}} \beta_{i,j} \geq 1, \text{ where } j \in \mathcal{S} \not\supset \mathcal{C}, \tag{1}$$

$$C_{i,neigh} \geq 1, \tag{2}$$

$$L^{act}_{i,j} = 1, \text{ where } i \in \mathcal{C} \text{ and } j \in \mathcal{S} \not\supset \mathcal{C} \tag{3}$$

where Equation (1) denotes that there exists at least one CS located at one-hop distance to which the traffic can be forwarded from an affected switch, $j \in \mathcal{S} \not\supset \mathcal{C}$. $\beta_{i,j}$ is a binary variable used to capture the availability of such CS, $i \in \mathcal{C}$, from an affected switch (AS), $j \in \mathcal{S} \not\supset \mathcal{C}$. Mathematically,

$$\beta_{i,j} = \begin{cases} 1, & \text{if packet forwarding is possible from} \\ & \text{AS, } j \in \mathcal{S} \not\supset \mathcal{C}, \text{ to CS, } i \in \mathcal{C} \\ 0, & \text{Otherwise} \end{cases} \tag{4}$$

Equation (2) denotes that number of neighbors of the CS, $i \in \mathcal{C}$, is greater than one, which, in turn, ensures that there exists at least another switch to forward the packet from the CS. Finally, Equation (3) ensures that the number of active links between an AS, $j \in \mathcal{S} \not\supset \mathcal{C}$, to CS, $i \in \mathcal{C}$, is one, so that redundancy of CS is avoided.

B. Greedy Heuristic Algorithm

The optimization problem presented in Section III-A is an integer linear programming (ILP) problem consisting of binary variables. Finding an optimal number of CS required to forward the traffic from an AS in the network is an NP-hard problem [13]. Consequently, we propose a greedy heuristic approach to solve the optimization problem in polynomial time. Algorithm 1 presents the proposed greedy heuristic algorithm. Time complexity of the proposed greedy algorithm consists of

Algorithm 1: Greedy Heuristic Algorithm

Input: Set of switches, $\mathcal{S}$; Adjacency matrix, $\mathcal{M}$, An array, A, with size $|\mathcal{S}|$
Output: Set of candidate switches, $\mathcal{C}$

```
Set |C| = φ, S_flag = 0 ∀S ∈ S;
Sort all the switches S ∈ S in descending order according to the number of their neighbors using M, and put them in A;
for i = 1 to |S| do
    if A[i]_flag == 0 && A[i]_neigh > 1 then
        C = C ∪ {A[i]};
        for j = i + 1 to |S| do
            if β_{A[i],j} == 1 then
                S_{l,flag} = 1;
Return C;
```

three phases – adjacency matrix formation, sorting and greedy algorithm. The running time complexities for adjacency matrix formation and sorting are $O(|\mathcal{S}|^2)$ and $O(|\mathcal{S}|log|\mathcal{S}|)$ having $|\mathcal{S}|$ number of switches in the network. The running time complexity for greedy approach is $O(|\mathcal{S}|^2)$. Therefore, total running time complexity of the proposed algorithm is $\big(O(|\mathcal{S}|^2) + O(|\mathcal{S}|log|\mathcal{S}|) + O(|\mathcal{S}|^2)\big) \Rightarrow O(|\mathcal{S}|^2)$.

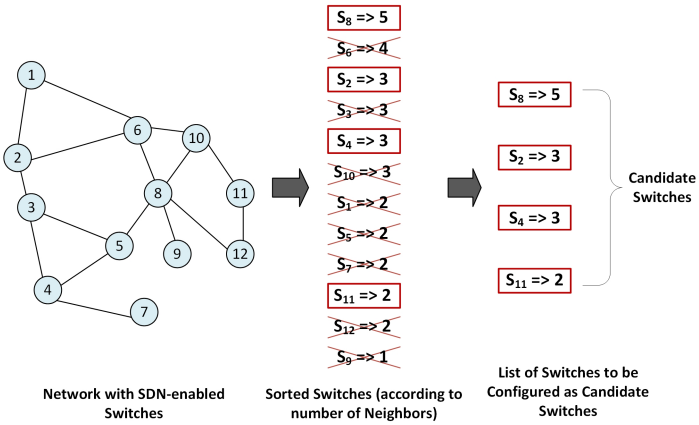

Fig. 4: Example: greedy heuristic approach for candidate switch selection for a given network

Example: Figure 4 shows an example of selecting candidate switches for a given network using greedy heuristic approach. In the example, part of the *Bandcon* [14] network topology is considered.

C. Tagging the PACKET-IN Message

We consider a modified version of OpenFlow protocol for sending PACKET-IN to the controller. Figure 5 shows different fields present in a PACKET-IN generated from an SDN switch according to OpenFlow protocol specification [15]. We use

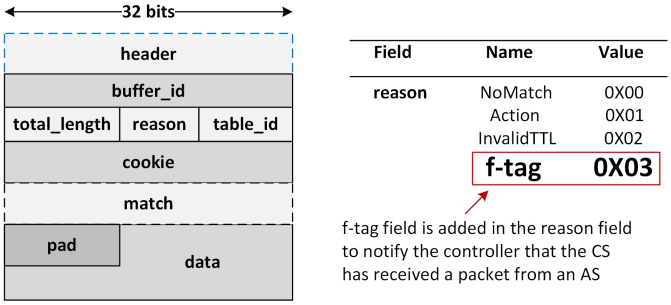

Fig. 5: PACKET-IN message sent to the controller

another **name** as **f-tag** in the **reason** field, and the **value** is set as **0X03** to notify the controller that a packet is received by a CS from an AS. It is noteworthy that the **f-tag** is used by the CS only. This method helps the controller to minimize the congestion at the AS further by placing the flow-rules in such a manner that new packets are not forwarded to the particular AS. It is noteworthy that the proposed approach can easily be integrated atop the existing SDN, in which OpenFlow is used as the communication protocol between the switch and the controller.

IV. Performance Evaluation

To evaluate the performance of the proposed scheme, we consider five different network topologies — *Arnes*, *Atmnet*, *AttMpls*, *Geant2012*, and *Goodnet* [14]. The number of nodes and links present in the networks are presented in Table I. Different parameters considered to conduct the experiment are summarized in Table II. We consider the rule-capacity of normal and candidate switches as 460 and 1526 according to the hardware support of HP S-2920 and HP S-3500 switches, respectively [16]. For flow-rule placement, we consider *exact-match* for all packets in the network. It is noteworthy that some of the packets have the same property, i.e., all fields in two different packets may be same. Therefore, a single flow-rule is applied for multiple packets in order to take desired action. We first determine the number of candidate switches required for a given network topology. Accordingly, we present the results for number of CS required for different network topology. Further, we evaluate the number of affected switches (AS) with different number of packets using the *AttMpls* network topology.

TABLE I: Network topologies considered for experiment [14]

Network Topology	**Number of Switches (bi-directional)**	**Number of Links**
Arnes	34	46
AttMpls	25	56
Atmnet	21	22
Geant2012	40	61
Goodnet	17	31

TABLE II: Simulation Parameters

Parameters	**Value**
Rule Capacity	460 (for HP S-2920) & 1526 (for HP S-3500) [16]
Source & Destination	Uniform Random
Number of Packets	22000 – 32000
Flow-Rule Placement	Exact-Match

To show the effectiveness of the proposed scheme, we evaluate the number of PACKET-IN and number of *congestion* scenarios in the network. It is noteworthy that we use the *AttMpls* network topology to present the results for PACKET-IN and congestion. Number of PACKET-IN denotes the total number of PACKET-IN received by the SDN controller. On the other hand, number of congestion scenarios represent the number of packets that experienced congestion at switches due to the rule-capacity constraints. We compared the performance of the proposed scheme with two different schemes — OpenFlow-based forwarding (OFS) and Randomized forwarding (RFS) [5]. In OFS, an existing rule is replaced by a new one,

if the rule-capacity of the switch is fully occupied. On the other hand, in RFS, the packet is forwarded to a randomly selected outgoing port without sending the PACKET-IN to the controller. Henceforth, we use the terms OFS and RFS to denote the existing schemes.

A. Number of Candidate and Affected Switches

We evaluate the number of candidate switches required to forward the traffic from an affected switch in the network according to the optimization problem and the proposed solution presented in Section III. Figure 6 shows the number of candidate switches required with different network topology. We see that the number of CS increases with an increase in

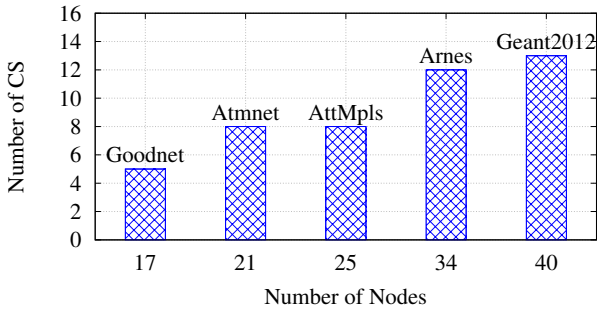

Fig. 6: Number of candidate switches with different network topology

the number of switches in the network. However, it remains unchanged for certain number of switches in the network. This is due to the fact that the number of required CS also depends on the number of links present in the network. For a dense network, number of CS is lower compared to a sparse network.

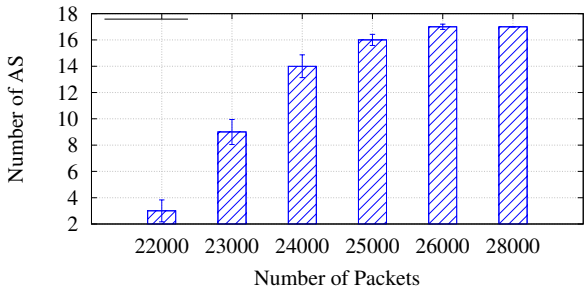

Fig. 7: Number of affected switches with different number of packets in AttMpls network

Further, Figure 7 shows the number of affected switches (AS) with different number of packets using AttMpls network. We see that the number of AS increases with an increase in the number of packets in the network. Finally, all the normal switches are affected with large number of packets in the network, and they forward traffic to the candidate switches.

B. PACKET-IN Message

Figure 8 shows the total number of PACKET-IN received by the controller with different number of packets in the network. We see that less number of PACKET-IN is sent to the controller using the proposed scheme, DynamiTE (proposed), and the existing scheme RFS. However, DynamiTE outperforms the OFS scheme in terms of the number of PACKET-IN sent to the controller. In DynamiTE and RFS, a switch does not send the message to the controller, if the rule-capacity is fully occupied. Therefore, the flow-rule associated with an active flow is not deleted. In contrast, the switch always sends the PACKET-IN on receiving a new packet without considering the residual rule-capacity. Consequently, more number of PACKET-IN is received by the controller using OFS than that of using DynamiTE (proposed) and RFS.

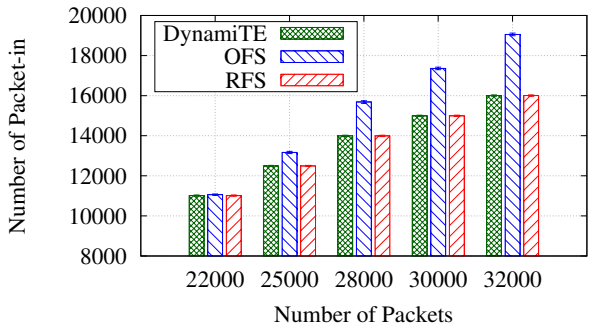

Fig. 8: Number of PACKET-IN with different number of packets in the AttMpls network

C. Congestion

Figure 9 denotes the number of packets that experienced congestion in the network using different schemes — DynamiTE (proposed), OFS and RFS. We see that the number of

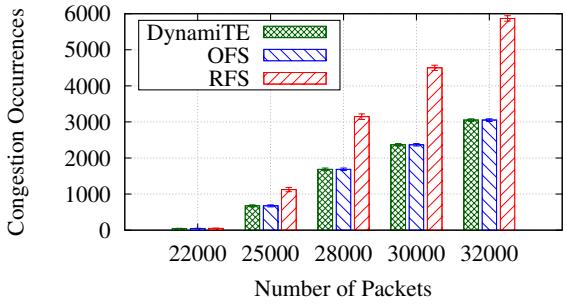

Fig. 9: Number of congestion occurred with different number of packets in the AttMpls network

packets that experienced congestion is the same for DynamiTE (proposed) and OFS. On the other hand, it is higher in case of RFS than that with the DynamiTE (proposed) and OFS schemes. In RFS, the packet is forwarded to a randomly selected switch, which can be further congested. Consequently, we have more number of congestion instances occurring using the RFS scheme.

Finally, we see that the proposed scheme, DynamiTE, outperforms the existing schemes — RFS and OFS — in terms

of the number of PACKET-IN and the number of congestion instances in the network, respectively. The proposed scheme is capable of reducing the number of PACKET-IN and congestion in the network by 10% and 38%, respectively.

V. Conclusion

In this paper, we proposed a dynamic traffic engineering scheme, DynamiTE, in an SDN-enabled network with an aim to minimize the number of PACKET-IN messages and congestion instances in the network. We proposed a greedy heuristic approach to determine optimal number of switches required to be configured as *candidate switches*. Further, we also proposed a packet-tagging approach to notify the SDN controller that rule-capacity of a switch is fully occupied. Through extensive simulation results, it is evident that the proposed scheme is capable of reducing the number of PACKET-IN received by the controller by 10%, compared to an OpenFlow-based forwarding scheme. Further, the proposed scheme is also capable of reducing the number of congestion instances occurring in the network due to rule-capacity constraint by 38%, compared to the existing randomized forwarding scheme.

The candidate switches have higher TCAM compared to the other switches in the network. However, rule-space of the candidate switches can also be fully utilized in the presence of large number of packets in the network. Therefore, some of the active flow-rules are required to be deleted from the candidate switches to insert new rules. Consequently, PACKET-IN messages are generated. We plan to analyze such overflow problem at the candidate switches as the future extension of this 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. Yassine, H. Rahimi, and S. Shirmohammadi, "Software defined network traffic measurement: Current trends and challenges," *IEEE Instrumentation & Measurement Magazine*, vol. 18, no. 2, pp. 42–50, Apr. 2015.

[3] S. Agarwal, M. Kodialam, and T. V. Lakshman, "Traffic Engineering in Software Defined Networks," in *Proceedings of the IEEE INFOCOM*, 2013, pp. 1–9.

[4] 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.

[5] S. Qiao, C. Hu, X. Guan, and J. Zou, "Taming the Flow Table Overflow in OpenFlow Switch," in *Proceedings of the ACM SIGCOMM*, Florianopolis, Brazil, Aug. 2016, pp. 591–592.

[6] M. Caria, A. Jukan, and M. Hoffmann, "A performance study of network migration to SDN-enabled Traffic Engineering," in *Proceedings of IEEE GLOBECOM*, Dec. 2013, pp. 1391–1396.

[7] R. Bhatia, F. Hao, M. Kodialam, and T. Lakshman, "Optimized network traffic engineering using segment routing," in *Proceedings of the IEEE INFOCOM*, 657–665, Apr–May 2015.

[8] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois, "The Segment Routing Architecture," in *Proceedings of IEEE GLOBECOM*, Dec. 2015, pp. 1–6.

[9] L. Davoli, L. Veltri, P. L. Ventre, G. Siracusano, and S. Salsano, "Traffic Engineering with Segment Routing: SDN-Based Architectural Design and Open Source Implementation," in *Proceedings of European Workshop on Software Defined Networks (EWSDN)*, Nov. 2015, pp. 111–112.

[10] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul, "FlowTags: enforcing network-wide policies in the presence of dynamic middlebox actions," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking (HotSDN)*, Hong-Kong, China, Aug. 2013, pp. 19–24.

[11] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying Middlebox Policy Enforcement Using SDN," in *Proceedings of the ACM SIGCOMM*, Hong-Kong, China, Aug. 2013, pp. 27–39.

[12] S. Bera, S. Misra, and M. S. Obaidat, "Mobility-Aware Flow-Table Implementation in Software-Defined IoT," in *Proceedings of the IEEE GLOBECOM*, Dec. 2016, pp. 1–6.

[13] R. M. Karp, *Reducibility among Combinatorial Problems*. Boston, MA: Springer US, 1972, pp. 85–103.

[14] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.

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

[16] P. Rygielski, M. Seliuchenko, S. Kounev, and M. Klymash, "Performance analysis of SDN switches with hardware and software flow tables," in *Proc, of the EAI International Conference on Performance Evaluation Methodologies and Tools (ValueTools)*, 2016, pp. 1–8.