QoS-Aware Adaptive Flow-rule Aggregation in Software-Defined IoT

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

***Abstract*—In this paper, we propose a QoS-aware adaptive flow-rule aggregation scheme in software-defined IoT (SDIoT) network with an aim to address flow-table overflow problem in SDN switches. The proposed scheme uses a key-based mechanism that is capable of fast aggregation and provides sufficient reduction in the number of flow rules, while having minimal impact on the QoS of IoT traffic. Further, we observe that it is necessary to adequately select a QoS path from multiple candidate paths, while considering the flow-table utilization at the switches. Accordingly, we present the *Best-fit* heuristic which takes into account the number of flow-rule insertions along with the bottleneck rule-capacity switch on a path, in order to minimize the total number of flow-rules in the network. Experimental results show that the proposed scheme is capable of reducing the average delay and packet drop by 35% and 12%, respectively, and improving the average throughput by 20% compared to the existing delay-based flow-aggregation scheme, while having comparable performance in terms of rule-aggregation.**

***Index Terms*—Software-Defined Networking, Quality of Service, Internet of Things, OpenFlow, Rule-compression**

I. Introduction

The evolution of Internet of Things (IoT) envisions a global interconnection of *billions* of smart objects such as laptops and smartphones, home appliances, and wireless sensor devices. With the number of smart networked devices projected to reach 27 billion by 2021 [1], sophisticated data and control networks are required to handle the massive data onslaught from devices that are extremely diverse in nature. Further, IoT systems are often geographically distributed and operate across a wide range of communication technologies. This suggests that IoT networks should be adaptable and flexible to changing operating conditions.

The software-defined networking (SDN) paradigm [2] precisely allows this flexibility by introducing programmability into the network. Recent advances in software-defined IoT (SDIoT) [3], [4] have shown the advantages of SDN in an IoT environment — in terms of simplification of network management, enabling diversified QoS for heterogeneous services, and resource utilization.

In SDN, the control logic is separated from data-plane forwarding and is relocated to a logically centralized SDN controller. This allows fine-grained control of individual flows using rule-based forwarding, based on optimal decisions by the centralized controller. Such fine-grained control is especially useful in satisfying the diverse QoS requirements of IoT — in terms of delay, jitter, packet-loss, and throughput

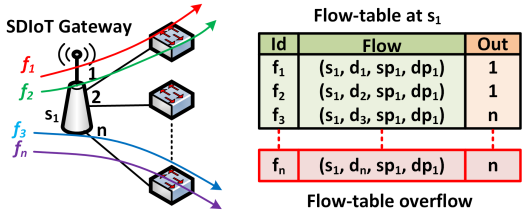


Figure 1: Example depicting flow-table overflow due to exact-match rules. Even though flows f_1, f_2 and f_3, f_n are forwarded out the same ports 1 and n, respectively, separate rules are placed for forwarding them due to exact-matching.

[5]. Rule-based forwarding in the data-plane is facilitated by OpenFlow, the *de-facto* standard in SDN [6]. The control logic and decisions of the centralized controller are translated by OpenFlow into simple *flow-rules* to be installed at the *flow-table* of switches. The flow-rules are in the form of match-action pairs, with each rule capable of matching on multiple fields such as ingress port, vlan id, ethernet, and tcp header fields. If a new flow matches any of the flow-rules in the flow-table, the corresponding action (such as output, modify, or drop) is applied. For every new flow that fails to find a match in the flow-table (table-miss), a *packet-in* message containing metadata about the flow is sent to the controller. After processing this message, an appropriate rule is placed in the flow-table at the switch to handle the new flow.

The ability of SDIoT to satisfy diverse QoS requirements arises from fine-grained control over individual flows, which allows the controller to take optimized decisions. However, fine-grained control using such flow-rules requires exact matching on multiple header fields, which, in turn, leads to a combinatorial increase in the number of flow-rules with large number of flows. On the other hand, the flow-table in SDN switches can accommodate only upto a few thousand entries, due to limitations of expense and power [7]. In an IoT scenario, where billions are devices are expected to be connected, this may lead *flow-table overflow* problem, as shown in Figure 1.

Recent studies [7]–[11] explored the flow-table overflow problem from different perspectives. Some of them [7], [8] focused on data-center networks (DCNs) in which proactively

placed wildcards[1] were used to reduce the number of flow-rules. Others [9]–[11] considered reactive strategies, in which flow-rules were aggregated after flow arrival. The existing solutions mostly focused on reducing the number of flow-rules without considering its impact on the QoS of individual network flows. The variety of IoT applications from diverse domains ranging from vehicular automation to healthcare require fine-grained application-specific QoS guarantees from the network [5]. Thus, it is necessary to consider the impact on QoS when considering flow-rule aggregation to address the flow-table overflow problem in SDIoT networks.

In this work, we attempt to address the question *"Is it possible to reduce flow-table overflow in an SDIoT network, while simultaneously preserving the QoS of IoT flows?"* We study the flow-table overflow problem in the SDIoT network and analyze the effect of existing flow-aggregation schemes on the QoS of IoT traffic. We observe that aggressive aggregation of flow-rules can lead to incorrect forwarding decisions which has an adverse effect on the QoS. Consequently, we propose a QoS-aware adaptive flow-rule aggregation scheme which offers sufficient reduction in the number of flow-rules while minimizing the impact on QoS of IoT traffic. The proposed scheme uses a key-based mechanism that is capable of fast aggregation of new flow-rules and leaves the choice of OpenFlow match-fields to the discretion of the user. We observe that a particular QoS path should be adequately chosen from multiple candidate paths while keeping in mind the flow-table utilization of the switches. Accordingly, we propose the *Best-fit* heuristic which considers the number of new flow-rule insertions, along with the bottleneck flow-table utilization on a path, in order to minimize the total number of flow-rules in the network.

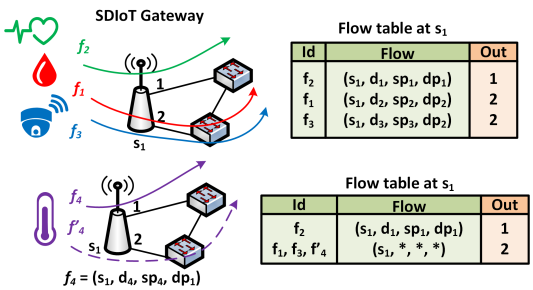
The rest of the paper is organized as follows. In Section II we analyze the relevant state-of-the-art. The problem statement is presented in III. Section IV presents the adaptive flow-rule aggregation scheme along with the proposed *Best-fit* heuristic. In Section V, we evaluate the performance of the proposed scheme. Finally, we conclude the paper in Section VI and present directions for future work.

II. Related Work

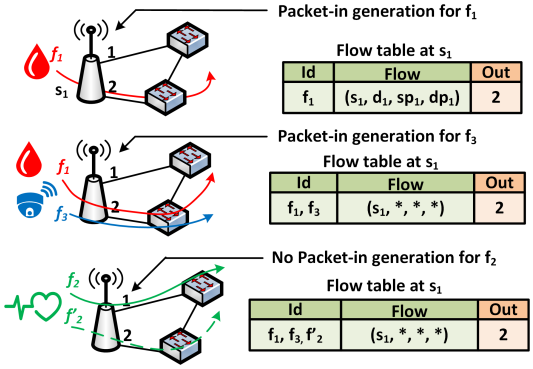
Kanizo *et al.* [7] and Kang *et al.* [8] addressed flow-table overflow in data-center networks (DCNs) by uniformly distributing flow-rules across the network. In the proposed schemes, the entire rule-space is divided into smaller sized tables and distributed across the network, while maintaining consistent routing policy at the controller. However, these are proactive schemes and require apriori knowledge about the characteristics of all flows. Thus, they are not suitable for the dynamic nature of an IoT environment.

Rifai et al. [9] proposed 'MINNIE', a heuristic approach for compressing flow-rules. The proposed approach consists of two phases — compression and routing. In the compression phase, flow-rules are aggregated using either source or destination and using the most occurring port. In the routing phase, the flow-table utilization is considered as a cost factor

[1]Wildcard flow-rules involve many header fields set to *don't care* bits.



((a)) Example where MINNIE [9] chooses incorrect forwarding path f'_4 for flow f_4 due to aggressive aggregation by srcip. At s_1, the correct output action for flows from s_1 with dstport dp_1 is out port 1. However, due to the $(s_1, *, *, *)$ rule, f_4 is forwarded incorrectly out port 2.



((b)) Example where [11] chooses incorrect forwarding path f'_2 for flow f_2, due to absence of packet-in message. Packet-in messages are generated for flows f_1 and f_3 due to table-miss. However, flow f_2 matches the aggregated flow rule $(s_1, *, *, *)$ and is forwarded incorrectly out port 2, before generation of packet-in message. Thus, the controller is unaware of flow f_2, and is unable to forward it along QoS path.

Figure 2: Example scenarios where the methods of [9] and [11] do not succeed, due to incorrect aggregation and reduced visibility (absence of packet-in message). In both the examples, exact-match flow rules are considered in the format (srcip, dstip, srcport, dstport, action). Correct forwarding paths are drawn solid, incorrect ones dashed.

in the routing of flows to achieve uniform usage of flow-tables across the network. As shown in Figure 2(a), compression using only source reduces the fine-grained forwarding capabilities of SDN. Moreover, in an IoT environment, it is important to consider QoS metrics such as delay, loss, jitter and throughput [5] as cost metrics for routing, instead of flow-table utilization.

Mimidis *et al.* [10] proposed a reactive flow-aggregation scheme for SDN-based backhaul networks, while considering the impact of aggregation on the QoS of flows. On receiving a packet-in message, the flows are separated into different QoS classes according to the IP differentiated services code point (DSCP) values. Then the controller checks if the recently arrived flow can be aggregated with existing flows while maintaining the QoS requirements. However, the authors did not present any details on *how* the flow-aggregation occurs. A similar flow-aggregation scheme was proposed by

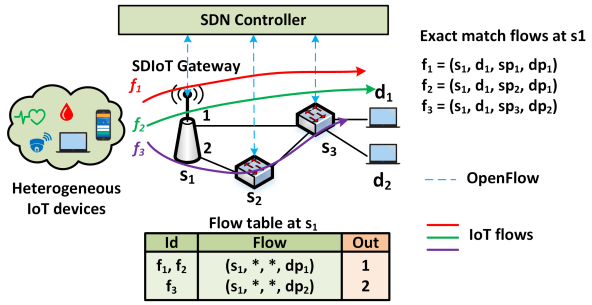


Figure 3: System Architecture

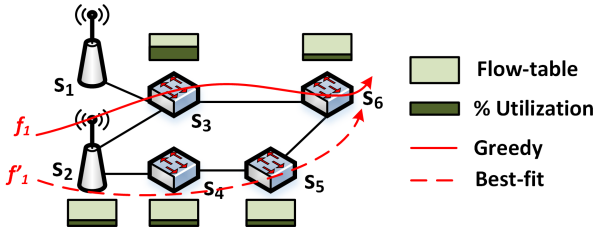


Figure 4: Example where naive greedy approach may lead to poor performance. The switch s_3 lies on multiple paths, thus, its flow-table gets filled up faster. The greedy approach chooses path f_1 with three new flow-rule insertions at s_2, s_3 and s_6. On the other hand, the Best-fit heuristic takes into account the bottleneck switch, s_3, and chooses path f_1' with four new flow-rule insertions at s_2, s_4, s_5 and s_6.

Kosugiyama *et al.* [11] based on total delay experienced by flows. Their approach is two-fold. First, on receiving a packet-in message, delay-constrained path is calculated for a flow, while minimizing the number of additional rules. Second, the flow-rules associated with the recently calculated path areaggregated into existing flow-rules, either along the entire path or partially. As shown in Figure 2(b), this approach may lead to the absence of packet-in message, which in turn, would lead to incorrect forwarding decisions.

Synthesis: The existing approaches have mainly focused on rule-aggregation without considering QoS of flows, or have considered QoS paths without analyzing the impact of flow-aggregation on the QoS paths. Thus, they are not suitable in an IoT environment. Detailed analysis of the literature suggests that there exists a research lacuna on the impact of flow-aggregation on the QoS of IoT flows. Thus, we present a QoS-aware flow-rule aggregation scheme for SDIoT network.

III. Problem Statement

A. Architecture

We consider an SDIoT architecture as shown in Figure 3. The heterogeneous IoT devices are connected to an SDN-enabled backbone through SDIoT gateways [12]. Let $\mathcal{S} = \{s_i \mid 1 \leq i \leq n\}$ denote the set of SDN-enabled switches. Each SDN-enabled switch, $s_i \in \mathcal{S}$, is also characterized by a flow-table, $\mathcal{R}_i$, consisting of a set of flow-rules. The SDN-enabled switches communicate with the centralized SDN controller through the OpenFlow protocol [6], and the traffic forwarding is controlled by placing forwarding rules in the flow-table at the switches.

We consider a flow-table rule r_j as a row-vector given as $r_j = \langle \mathcal{M}_j, \mathcal{A}_j, \mathcal{C}_j \rangle$, where $\mathcal{M}_j$, $\mathcal{A}_j$ and $\mathcal{C}_j$ represent the set of match-fields, actions, and flow-counters associated with the j^{th} rule, respectively. The match-field set, $\mathcal{M}_j$, associated with the j^{th} rule, is defined as $\mathcal{M}_j = \{m_{j,i} \mid 1 \leq i \leq n\}$ where n is the number of match-fields, which varies from 12 to 44 according to the OpenFlow specification. Therefore, the flow-table associated with a particular switch, $s_i \in \mathcal{S}$, is given as $\mathcal{R}_i = \{r_j^i \mid 1 \leq j \leq \mathcal{R}^{max}\}$, where $\mathcal{R}^{max}$ denotes the maximum number of rules in a flow-table.

B. Presence of Heterogeneous Flows

The heterogeneity of devices and networking resources in IoT necessarily implies the existence of heterogeneous traffic types in the network, with differentiated QoS requirements — in terms of delay, jitter, packet-loss, and throughput [5]. Some IoT applications such as factory and process automation may be *latency-critical* [13], while applications such as environmental monitoring may only require periodic updates and can tolerate a reasonable amount of delay. Thus, fine-grained application-specific QoS treatment is required for IoT traffic, using state-of-the-art SDN-based schemes such as [5].

However, using exact-match on all OpenFlow headers to realize such fine-grained QoS treatment may lead to flow-table overflow, as explained in Section I. On the other hand, aggregating flow-rules aggressively, using fixed match-fields such as *only* source, reduces the network visibility and may lead to incorrect forwarding decisions, as shown in Figures 2(b) and 2(a). Therefore, a suitable combination of match-fields needs to be chosen as the basis for flow-aggregation, in order to adequately satisfy the QoS of IoT flows. For example, from Figure 3, we observe that aggregating the flow-rules using a combination of source and destination port i.e., $(s_1, *, *, dp_1)$ is capable of correctly forwarding the IoT flows under consideration (also refer to Figure 2(a)). Further, since IoT traffic is usually low-rate, multiple flows can be aggregated using a wildcard rule such as $(s_1, *, *, dp_1)$, as long as $(s_1, *, *, dp_1)$ is enough to differentiate between IoT flows requiring different QoS treatment.

C. Adaptive flow-rule aggregation

As flows arrive sequentially, multiple candidate paths may be present that satisfy the QoS requirements of each flow. Therefore, from the candidate paths, a particular path has to be suitably chosen keeping in mind the flow-table constraints. A naive approach would be to simply pick the path with minimum number of new flow-rule insertions. However, such an approach may lead to faster fill up of a bottleneck switch as shown in Figure 4. If the bottleneck switch, s_3, is completely filled, new flows arriving at that switch will be dropped. Consequently, all paths with s_3 as an intermediate switch will become invalid, leading to sub-optimal performance. Therefore, we propose an adaptive heuristic, termed *Best-fit*, to address this issue.

IV. Solution Approach

In this Section, we present the *Best-fit* heuristic and key-based rule-aggregation mechanism for adaptive flow-rule aggregation.

Definition 1. Best-fit heuristic : Given a set of paths[2], choose the path P with minimum cost $\delta(P)$. The cost of choosing a path P is given as

$$\delta(P) = \sum_{s_i \in P} \left(\alpha\lambda + \beta \max_i \left(\frac{|\mathcal{R}_i|}{\mathcal{R}^{max}} \right) \right) \quad (1)$$

where λ represents the cost of inserting a new flow-rule and α, β are normalizing constants.

The Best-fit heuristic takes into account the cost of adding new flow-rules along with the flow-table utilization, $\mathcal{R}_i/\mathcal{R}^{max}$, at each switch, as shown in Figure 4. Therefore, it does not suffer from adverse effects of bottleneck switches.

Algorithm 1 Adaptive path selection

Inputs: Set of OpenFlow switches, $\mathcal{S}$, Set of flows, $\mathcal{F}$
Output: Aggregated flow-rules for the set of flows

1: **for** each $s_i \in \mathcal{S}$ **do**
2: initialize empty dictionary, d_i
3: **for** each flow $\in \mathcal{F}$ **do**
4: get set of QoS paths, $\mathcal{P} = \{P_k \mid 1 \leq k \leq |\mathcal{F}|\}$ using existing schemes such as [5].
5: get the least cost path using the **Best-fit** heuristic
6: place aggregated flow-rules along least cost path using **Algorithm 2**

Algorithm 1 is used to adaptively select a QoS path for a flow based on the Best-fit heuristic, in order to minimize the total number of flow-rules in the network. The SDN controller maintains a dictionary, d_i, for each switch $s_i \in \mathcal{S}$ in order to store the keys based on which flow-aggregation takes place. For each new flow (packet-in) at the SDN controller, the set of QoS paths is analyzed using the Best-fit heuristic and the one with the minimum cost according to Equation (1) is chosen. Calculating the cost $\delta(P)$ for path P involves two operations — checking the number of new rule insertions and calculating the bottleneck flow-table utilization. The bottleneck flow-table utilization can be calculated by checking the flow-table utilization of each switch $s_i \in P$ in $\mathcal{O}(|P|)$ time. To check whether a new flow-rule will be placed at a particular switch $s_i \in P$ involves extracting the key (refer to Step 5 of Algorithm 2) and checking in constant time[3] whether the key exists in the dictionary d_i. Thus, the complexity of calculating the cost of path P is $\mathcal{O}(|P|)$, where $|P|$ is the length of the path. Assuming that a state-of-the art QoS scheme for IoT such as [5] returns at most k QoS paths, the complexity of flow-rule aggregation for a particular flow is $\mathcal{O}(k|P|)$ and is therefore, linear.

[2]A path is an ordered sequence of OpenFlow switches.
[3]Average key lookup time in hash-based dictionary is $\mathcal{O}(1)$

Algorithm 2 Flow aggregation

Inputs: QoS path, P_k, for the k^{th} flow, set of switches, $\mathcal{S}$, and set of user-defined match-field indices, $\mathcal{I} \subset \{1, 2, \cdots, n\}$, based on which aggregation takes place.
Output: Aggregated flow rules placed along path P_k

1: **for** each $s_i \in P_k$ **do**
2: get exact-match rule r_k
3: AGGREGATE($r_k, s_i, \mathcal{I}$)
4: **procedure** AGGREGATE($r_k, s_i, \mathcal{I}$)
5: extract key λ_k from r_k such that $\lambda_k \leftarrow m_{k,p}, m_{k,q}, \cdots, m_{k,t}, a_k$, where $p, q, \cdots, t \in \mathcal{I}$
6: **if** $\lambda_k \in keys(d_i)$ **then**
7: append value $d_i[\lambda_k] \leftarrow r_k$
8: modify rule r_k to $r'_k = \langle \mathcal{M}'_k, \mathcal{A}'_k, \mathcal{C}'_k \rangle$ such that $m'_{k,i} \leftarrow m_{k,i} \quad \forall i \in \mathcal{I}, \quad m'_{k,i} \leftarrow * \quad \forall i \notin \mathcal{I}$, $\mathcal{A}'_k \leftarrow \mathcal{A}_k$ and $\mathcal{C}'_k \leftarrow \mathcal{C}_k$
9: update rule r'_k in flow-table of s_i
10: **else if** $\lambda_k \notin keys(d_i)$ **then**
11: insert key-value pair $d_i[\lambda_k] \leftarrow r_k$
12: place r_k in flow-table of s_i

The proposed key-based flow rule-aggregation scheme is presented in Algorithm 2. For each flow-rule, r_k, in a flow-table, a unique key, λ_k, is created based on a set of user-defined match-field indices, $\mathcal{I}$, where $\mathcal{I}$ is a subset of all the match-field indices in a flow-rule (Step 5). The dictionary d_i associated with each switch is checked for the existence of the key λ_k (Step 6). If the key is found, we set the match-fields which are not specified by $\mathcal{I}$, as *don't-care* (*) to get aggregated rule, r'_k (Steps 7 to 9). The complexity of Algorithm 2 is O(1), since it involves only key-based dictionary look-ups. Thus, Algorithm 2 is capable of aggregating the flow rules with minimal additional delay.

V. Performance Evaluation

A. Simulation Settings

We evaluated the performance of the proposed scheme using the POX SDN controller[4] and the Mininet network emulator [14]. For all the experiments, we considered a scale-free topology using the Barabesi-Albert model [15]. We used the D-ITG traffic generator to model IoT traffic flows based on real traces in [16]. The different simulation parameters are presented Table I.

B. Benchmarks

To show the effectiveness of the proposed scheme, we compare the proposed scheme (Best-fit heuristic) with the following baselines — existing delay-based flow-aggregation scheme (Agg-Delay) [11], flow-aggregation using Random and Greedy heuristic, and no aggregation (Exact-match). For flows having same QoS path, Agg-Delay aggregates flow rules using only the source. The Random, Greedy, and the proposed scheme consider a combination of source

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

Table I: Simulation parameters

Parameter	Value
Number of switches	10
Number of links	25
Maximum rules at a switch	200
Avg. packet size	94 – 699 bytes [16]
Active volume	142 – 27,716 bytes [16]
Mean rate	562 - 516,540 bps [16]
Active time	1 – 34 s [16]

and destination port as the key for flow aggregation. From multiple QoS paths, Greedy selects the path with minimum number of new flow-rule insertions and Random selects a path randomly, while the proposed scheme selects the path using the *Best-fit* heuristic.

C. Performance Metrics

We consider the following performance metrics to evaluate the performance of the proposed scheme.

(i) *Average delay*: The average end-to-end delay experienced by flows in the network. This includes the network delay and the flow-setup delay experienced by the flows at the OpenFlow switches.

(ii) *Packets dropped*: The number of packets dropped in the network. This includes loss due to incorrect forwarding decisions as well as loss due to congestion.

(iii) *Average throughput*: The average data rate delivered to all end-hosts in the network.

(iv) *Reduction in flow-rules*: The percentage reduction in flow-rules at the switches due to flow-rule aggregation as compared to exact-match rules (no aggregation).

D. Results and Discussion

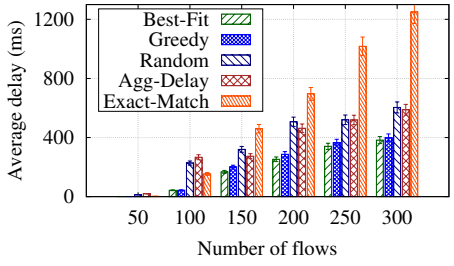


Figure 5: Average end-to-end delay with different number of flows in the network

Figure 5 shows the average end-to-end delay with different number of flows. We observe that the proposed scheme (Best-fit) outperforms the Agg-Delay and Exact-match schemes in all the cases, while achieving comparable performance with the Greedy scheme. In particular, with 300 flows in the network, the proposed scheme achieves 35% and 70% reduction in end-to-end delay compared to the Agg-Delay and Exact-match schemes, respectively. The Exact-match scheme performs the worst and incurs significantly higher delay compared to the other schemes. This is due to the fact that no aggregation occurs in the Exact-match scheme, and all flows incur significant flow-setup time[5] at the SDN controller. The Agg-Delay scheme incurs higher delay than the proposed scheme because flows which do not generate packet-in at the controller (refer to Figure 2(b)) may take sub-optimal route to destination due to aggregation.

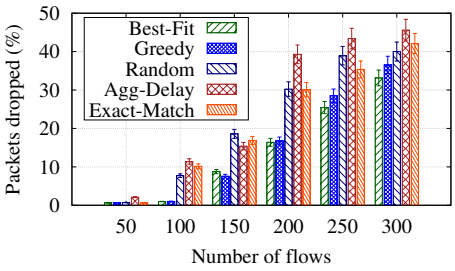


Figure 6: Packets dropped with different number of flows in the network

Figure 6 shows the percentage of packets dropped with different number of flows in the network. We observe that the proposed scheme (using Best-fit) outperforms the benchmark schemes. In particular, we observe that the proposed scheme incurs 4%, 7%, 10%, and 12% less packet drop compared to the Greedy, Random, Exact-match, and Agg-Delay schemes, respectively. The Agg-Delay scheme incurs higher packet drop as it aggressively aggregates flow-rules using only source, which limits the network visibility and can lead to incorrect forwarding decisions (as explained in Figure 2(b)). Surprisingly, the proposed scheme also outperforms the Exact-match scheme in terms of packets dropped. This is due to the excess delay experienced by flows in the Exact-match scheme (refer to Figure 5), which leads to violation of QoS constraints. Further, we observe that with higher number of flows in the network, the proposed scheme incurs less packet-drop than the Greedy scheme. This is due to the Best-fit heuristic which considers the bottleneck flow-table utilization on a path so that QoS paths are not invalidated by one particular switch on the path suffering from flow-table overflow, as compared to the other switches.

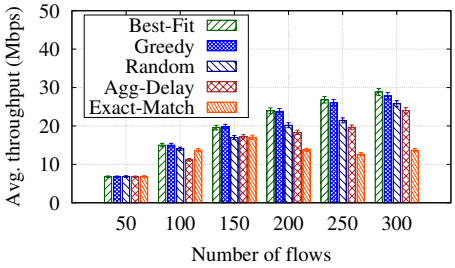


Figure 7: Average throughput in the network

Figure 7 shows the average throughput in the network with varying number of flows. We observe that the proposed

[5]The time taken by the SDN controller to install the corresponding flow-rule at the SDN switch. For example, it takes ∼100 ms for installing 100 flow-rules in the HPJ9538A switch.

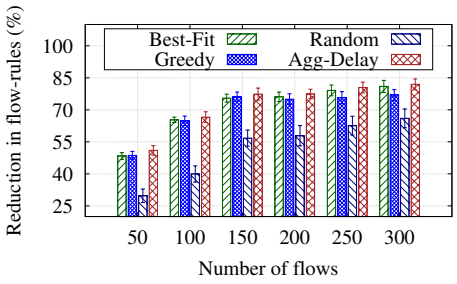


Figure 8: Reduction in flow-rules (averaged over all switches) with different number of flows

Figure 9: Reduction in flow-rules at each switch with 200 flows in the network

scheme outperforms the benchmark schemes in all the cases. In particular, with 300 flows in the network, the proposed scheme offers 4%, 11%, 20% and 110% improvement in average throughput compared to the Greedy, Random, Agg-Delay, and Exact-match schemes, respectively. The average throughput depends on the packet drop and the average delay in the network. Consequently, the Exact-match scheme achieves significantly less throughput due to its high end-to-end delay and packet-loss.

Figures 8 and 9 show the percentage of reduction in flow rules compared to the Exact-match scheme. We observe that the proposed scheme achieves similar performance to the Agg-Delay and Greedy schemes in terms of flow-rule aggregation. It is noteworthy that Agg-Delay considers only the source field for aggregation, while the proposed scheme (Best-fit) and Greedy scheme consider a combination of source and destination port as key for aggregation. Figure 9 shows the percentage of reduction in flow rules at each switch compared to the Exact-match scheme. We observe that the proposed scheme (Best-fit) achieves more uniform reduction in flow-rules across all switches, as compared to the Random and Greedy scheme. Consequently, the proposed scheme does not suffer from flow-table overflow at bottleneck switch as explained in Section III-C.

VI. Conclusion

In this paper, we proposed an QoS-aware adaptive flow-rule aggregation scheme for software-defined IoT networks with an aim to address the problem of flow-table at SDN switches. The proposed scheme utilizes an adaptive key-based aggregation mechanism along with the *Best-fit* heuristic to minimize the total number of flow-rules in the network, while having minimal impact on the QoS of IoT flows. In particular, with 300 flows in the network, the proposed scheme is capable of reducing the average delay and packet drop by 35% and 12%, respectively, and improving the average throughput by 20% compared to the existing delay-based flow-aggregation scheme, while having comparable performance in terms of rule-aggregation.

In this work, we only considered the existence of low-rate IoT traffic in the network. However, in the ubiquitous IoT network, such low-rate IoT traffic will co-exist with traditional Internet traffic, which will affect the QoS. Thus, as the future extension of this work, we aim to incorporate adequate mechanisms to handle such traffic while preserving flow rule-aggregation.

References

[1] Cisco Systems Inc., "The Zettabyte Era: Trends and Analysis," *White Paper, Cisco Visual Networking*, 2014.

[2] D. Kreutz, F. M. V. Ramos, P. E. Verssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[3] N. Bizanis and F. A. Kuipers, "SDN and Virtualization Solutions for the Internet of Things: A Survey," *IEEE Access, Doi: 10.1109/ACCESS.2016.2607786*, 2016.

[4] K. Sood, S. Yu, and Y. Xiang, "Software-Defined Wireless Networking Opportunities and Challenges for Internet-of-Things: A Review," *IEEE Internet of Things J.*, vol. 3, no. 4, pp. 453–463, 2016.

[5] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian, "A Software Defined Networking architecture for the Internet-of-Things," in *Proc. IEEE Network Operations and Management Symposium (NOMS)*, 2014, pp. 1–9.

[6] 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 CCR*, vol. 38, no. 2, pp. 69–74, 2008.

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

[8] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the "One Big Switch" Abstraction in Software-defined Networks," in *Proc. ACM CoNEXT*, 2013, pp. 13–24.

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

[10] A. Mimidis, C. Caba, and J. Soler, "Dynamic aggregation of traffic flows in SDN: Applied to backhaul networks," in *Proc. IEEE NetSoft*, 2016, pp. 136–140.

[11] T. Kosugiyama, K. Tanabe, H. Nakayama, T. Hayashi, and K. Yamaoka, "A flow aggregation method based on end-to-end delay in sdn," in *Proc. IEEE ICC*, 2017, pp. 1–6.

[12] A. Hakiri, P. Berthou, A. Gokhale, and S. Abdellatif, "Publish/subscribe-enabled Software Sefined Networking for Efficient and Scalable IoT Communications," *IEEE Commun. Mag.*, vol. 53, no. 9, pp. 48–54, 2015.

[13] P. Schulz, M. Matthe, H. Klessig, M. Simsek, G. Fettweis, J. Ansari, S. A. Ashraf, B. Almeroth, J. Voigt, I. Riedel, A. Puschmann, A. Mitschele-Thiel, M. Muller, T. Elste, and M. Windisch, "Latency Critical IoT Applications in 5G: Perspective on the Design of Radio Interface and Network Architecture," *IEEE Commun. Mag.*, vol. 55, no. 2, pp. 70–78, 2017.

[14] 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, pp. 19:1–19:6.

[15] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[16] 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. IEEE INFOCOM Workshops*, 2017, pp. 559–564.