Dynamic Network Slice Assignment in Software-Defined IoT Networks

Niloy Saha and Sudip Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur, 721302, India
Email: niloysaha@iitkgp.ac.in, sudipm@iitkgp.ac.in

***Abstract*—Network slicing using technologies such as software-defined networking is a promising approach to address the diverse quality-of-service (QoS) requirements of Internet of Things (IoT) applications. Network slicing allows a single physical network to be transparently divided into separate, logically independent networks, with distinct service levels per slice. This allows the network to support the diverse requirements of different IoT verticals ranging from connected vehicles to smart agriculture. In this paper, we propose a dynamic network slice assignment scheme for software-defined IoT networks, where each application type is assigned to a distinct *slice* controller depending on its QoS requirements. We formulate the optimal mapping between network slices and slice controllers as an integer programming problem, and show that the proposed scheme is capable of significantly reducing the number of QoS violations compared to existing controller mapping schemes. Further, we present a scheduling scheme to dynamically re-compute the optimal mapping, based on optimal stopping theory, while taking into consideration the temporal variations in traffic and link latencies. Simulation results show that the proposed scheduling strategy is able to reduce the number and expected cost of re-computations, while remaining within the QoS violation tolerance limit.**

***Index Terms*—Internet of Things, Network Slicing, Software-Defined Networking**

I. Introduction

The Internet of Things (IoT) encompasses a massive variety of use-cases, ranging from smart agriculture to connected vehicles, which require diverse quality of service (QoS) from the network. Next-generation networking technologies such as software-defined networking (SDN) have shown significant promise in addressing these requirements, by simplifying network management, and making the network flexible [1]. In particular, the concept of network slicing allows the creation of multiple virtual networks (*slices*), customized for particular applications, on top of a given physical network [2]. These virtual networks offer mutual isolation, and independent control and management, which makes them particularly well suited to address the diverse requirements of IoT applications. Network slicing using SDN allows each slice to be controlled independently by a separate SDN controller [3], [4]. Thus, by creating a separate slice for each application, the SDN control application can be tailored to meet application-specific QoS requirements of IoT [5].

An SDN controller provides fine-grained control over flows using instructions in the form of match-action flow-rules at the switches. On flow arrival at a switch, if a match is not found, a request is sent to the controller, which then places the appropriate flow-rule(s) to handle the flow. Thus, the controller response time plays a significant role in the delay experienced by the flows. This implies that the delay experienced by a network slice depends on the resource capacity of the associated slice controller[1]. Machine-to-machine (M2M) applications are expected to be a large part of IoT [6], where some applications are mission-critical and have stringent latency requirements, such as connected vehicles and smart healthcare, while others can tolerate some amount of delay, such as smart agriculture [7]. Thus, the network slices must be mapped to the slice controllers while taking into consideration the distinct application-specific requirements, and the resource capacities of the controllers.

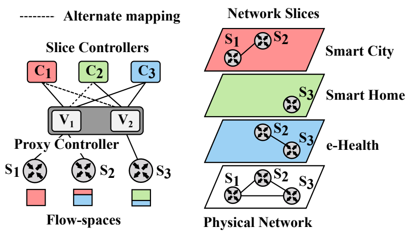

Fig. 1: SDN-based network slicing for IoT

Figure 1 shows the concept of network slicing using SDN, with each IoT application allocated a separate network slice. A proxy controller is used to logically partition the address space (of flow-rules) into separate flow-spaces. Each flow-space represents the address range of flows belonging to a distinct network slice and is used to control all traffic belonging to that slice. Thus, an SDN controller manages a network slice using its corresponding flow-space. Figure 1 (right) shows the virtual topologies associated with each network slice, with each slice controlling traffic from a particular IoT application. In the figure, the colored boxes below the switches s_1, s_2, and s_3 represent the number of requests belonging to a particular flow-space, generated from each switch. From the figure, we observe that there are several choices for assigning a flow-space to a slice controller. For example, as opposed to the mapping shown in the figure (shown with solid lines), the blue flow-space could have been mapped to controller c_2, while controller c_3 controlled the green flow-space. Note that

[1]Since the controller response time is dependent on its resource capacity.

to get uniform treatment for all flows belonging to a network slice, the corresponding flow-space has to be mapped to a *single* controller, i.e., fractional allocations are not permitted. The problem of selecting the appropriate mapping between flow-spaces and controllers, while considering the dynamic network conditions, and requirements of the IoT applications is challenging. Thus, in this paper, we propose a flow-space to controller mapping (FCM) scheme to address this issue. Additionally, the temporal variation in IoT traffic and link latencies in IoT networks can cause the optimal mapping to become sub-optimal over time, in which case the mapping must be re-computed. Therefore, we propose a dynamic scheduling strategy to re-compute the mapping, so that the expected cost of such re-computation is minimized.

The rest of the paper is structured as follows. In Section II, we discuss the existing literature. In Section III we present the optimal flow-space to controller mapping problem. The dynamic scheduling strategy is presented in Section IV. We analyze the performance of the proposed scheme in Section V, and finally conclude the paper in Section VI.

II. Related Work

We analyze the relevant state-of-the-art from two perspectives — network slicing, and optimal controller mapping.

Network Slicing using SDN. Recent works [2], [8] identify SDN as a key enabler for network slicing in IoT and 5G networks. Sherwood *et al.* [3] proposed Flowvisor, a framework for network slicing using OpenFlow, the *de-facto* protocol for SDN. In Flowvisor, a transparent proxy controller was introduced between the SDN switches and controllers, in order to slice various network resources, such as address space, bandwidth, and CPU. This concept was extended by Salvadori *et al.* [4] to enable the creation of generic bandwidth guaranteed virtual topologies, on top of a shared physical network. Chaabnia and Meddeb [9] showed how network slicing could be used to provide differential QoS in software-defined home networks. The authors categorized the applications into different classes, and each class was assigned to a specific network slice, created using Flowvisor. Similar to [4], [9], in our work, we adopt the Flowvisor concept for creating network slices using flow-spaces.

Optimal Controller Mapping. Recent works [10]–[13] proposed dynamic controller mapping in SDN for improved resource utilization. Wang *et al.* [10] studied dynamic mapping between SDN switches and controllers by formulating the optimal assignment problem as a matching game. The authors showed that the matching-based approach reduced the average response time and total overhead in the network. However, they considered one-to-one mapping between switches and controllers, and did not consider network slicing. Sridharan *et al.* [11] studied fractional mapping between switches and multiple controllers. The authors considered that the switches were simultaneously connected to multiple controllers, and a fraction of the flow-requests generated at a switch was sent to each controller. The authors showed that fractional mapping was capable of more efficient load balancing, in comparison to one-to-one mapping. AL-Tam and Correia [12] also studied fractional mapping between switches and controllers; however, they focused on minimizing the number of new assignments (state changes) under dynamic network conditions. In our recent work [13], we showed how SDN network slicing could be leveraged to minimize the overall controller response time, with distributed (but not independent) SDN controllers. The proposed scheme differs from the existing works in two aspects. First, we focus on the optimal mapping between network slices and SDN controllers. Thus, each controller is responsible for traffic from exactly one flow-space. This is in contrast to the fractional mapping schemes proposed in [11], [12], where a controller serves traffic from all applications, and allows us to model the controller response time for a particular application. Second, we propose a dynamic scheduling strategy to re-compute the controller assignment, so that the expected cost of re-computation is minimized.

III. Optimal Flow-space Mapping

A. System Model

We consider a virtualized SD-WAN network, with distributed proxy controllers and multiple slice controllers, as shown in Figure 1. Let $C, V, S,$ and F denote the set of SDN slice controllers, set of proxy controllers, SDN-enabled switches, and flow-spaces, respectively. Further, let L denote the set of links between the switches, with each link $l \in L$ having an associated propagation delay δ_l. Aggregated IoT traffic from various heterogeneous networks arrive at the switches S, after passing through IoT gateways [5]. Although the traffic from individual IoT devices may be bursty or periodic in nature, the aggregated traffic from different sources arriving at a switch can be approximated as a Poisson process [14]. After arrival at a switch, if a flow does not find any matching flow-rule(s), a request is sent to the controller in the form of a *packet-in* message. Packets belonging to the flow can only be forwarded once the packet-in request has been processed and an appropriate flow-rule has been placed at the switch. Thus, the delay experienced by a flow consists of — a) packet-in generation delay at the switch, b) propagation delay from switch to controller, c) packet-in processing delay at the controller, and d) delay incurred in placing the appropriate flow-rule at the switch. Apart from this, the intermediate virtualization layer in the proposed scheme also adds a non-negligible constant overhead. The delay factors (a) and (d) can be significantly reduced using various existing schemes [15], and are independent of the controller association. Thus, we use a constant term Δ^{net} to represent the net delay contributed by packet-in, proxy overhead, and flow-rule placement, and focus mainly on the two delays affected by controller association — propagation delay, and packet-in processing delay. The proxy controller V acts as an intermediary between the SDN slice controllers and the data-plane switches. To avoid a single point of failure, the proxy controller is based on a distributed SDN controller. The different functional components of the proxy controller are shown in Figure 2.

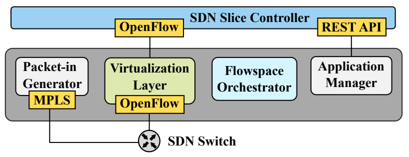

Fig. 2: Overview of the proxy controller

The *virtualization layer* is based on the Flowvisor concept proposed in [3], [4], and is used to slice the SDN network using the OpenFlow protocol. It virtualizes the address space at SDN switches into mutually independent flow-spaces, which allows a slice controller to have independent control over all flows belonging to a particular flow-space. The *flow-space orchestrator* is responsible for the dynamic mapping between the flow-spaces and slice controllers. It collects the necessary network statistics (such as link latencies, and incoming request rate) by interfacing with the virtualization layer, and runs the algorithm to solve the FCM problem. Since each slice controller is dedicated to a particular IoT use-case, the control application running on them may be specifically tailored to address the characteristics and requirements of that use-case. The *application manager* component of the proxy controller is responsible for copying the necessary application binaries for a particular IoT application to the appropriate slice controller using a REST API. Note that the application binaries may be stored on the proxy controller itself, or may be pulled in from a remote server. We also introduce a *packet-in generator* component in the proxy controller, connected to the switches using a short label-switched path. This is used to offload the packet-in generation task from the limited switch CPUs to the more capable proxy controller, and thus, significantly reduce the delay of packet-in generation [15]. This reduction in the delay is used to offset the additional overhead of the virtualization layer. We leave the analysis of the net delay of the virtualization layer, as future work.

The Poisson arrivals at the switches imply that the probabilistic flow-rule miss and subsequent packet-in generation rate is also Poisson [16]. Let $\lambda_s^f(t)$ be the packet-in generation rate at switch $s \in S$, belonging to flow-space $f \in F$, in time period t. We consider discrete time intervals t which allows us to accurately measure packet-in requests. The packet-in requests from s are sent to the distributed proxy controller instance $v \in V$, which is associated with switch s. For simplicity, we consider the mapping x_{sv} between switch s and proxy controller v to be static, following a simple load-balancing scheme[2], as used in popular SDN controllers, such as ONOS[3]. However, the proposed scheme is also applicable to dynamic mapping. Therefore, at v, the total request rate belonging to flow-space f is given as $\gamma_v^f(t) = \sum_s \lambda_s^f(t) x_{sv}$.

B. Optimization Problem for Optimal Mapping

1) Decision variables: We consider binary variables $y_{vc}^f(t)$ to denote whether flow-space $f \in F$ from proxy controller $v \in V$ is mapped to slice controller $c \in C$ in time period t. We also consider the auxiliary binary variable $g_c^f(t)$ to denote whether flow-space f is mapped to slice controller c, irrespective of the proxy controller instance. Mathematically, $g_c^f(t) = 1$, if $\sum_v y_{vc}^f(t) > 0$, and $g_c^f(t) = 0$, otherwise.

2) Objective: Our objective is to find the appropriate mapping of flow-spaces to the slice controllers, such that the control overhead and service cost is minimized, while satisfying the requirements of the IoT applications. The slice controllers C may be connected to the proxy controller instances V through various multi-hop paths. The asynchronous packet-in messages traversing these multi-hop paths generate control overhead, that should be kept to a minimum. The request arrival rate at controller c from proxy v is given as $\theta_{vc}(t) = \sum_f \gamma_v^f(t) y_{vc}^f(t)$. Therefore, the control overhead at slice controller c for time period t is given as $\phi_c^{ctrl}(t) = \sum_v h_{vc}\theta_{vc}(t)$, where the term h_{vc} denotes the hop-count between v and c. Let ψ_c^{serv} be the unit cost per request for using slice controller c, which includes operating expenses, such as energy and maintenance costs. The total request arrival rate at slice controller c, from all proxy controllers, is given as $\theta_c(t) = \sum_v \theta_{vc}(t)$. Therefore, the service cost for using slice controller c during time period t, is given as $\phi_c^{serv}(t) = \psi_c^{serv}\theta_c(t)$.

3) Constraints: There is a one-to-one mapping between flow-spaces $f \in F$ and slice controllers $c \in C$. Mathematically:

$$\sum_{f\in F} g_c^f = 1 \quad \forall c \in C, \quad \sum_{c\in C} g_c^f = 1 \quad \forall f \in F \tag{1}$$

Further, the request rate at a slice controller c should be less than its service rate μ_c. Mathematically,

$$\theta_c(t) \le \beta\mu_c \quad \forall c \in C \tag{2}$$

where the term $\beta \in (0, 1)$ represents the maximum allowed utilization of the processing resources at c. We consider an in-band communication model, where the data and control traffic share the same links $l \in L$. The rationale behind this is that a dedicated control network (out-of-band) over long distances would incur significant deployment costs in a WAN. Thus, the propagation delay from proxy controller v to slice controller c is given as $\Delta_{vc}^{prop} = \sum_{l\in P_{vc}} \delta_l$, where P_{vc} denotes the shortest path from proxy controller v to slice controller c. Further, considering the slice controller as a M/M/1 queue, and using Little's theorem, the average response time (including waiting and processing time) of a packet-in message at slice controller c is given as $\Delta_c^{proc}(t) = 1/(\mu_c - \theta_c(t))$. Therefore, the maximum delay experienced by an IoT application (represented by flow-space f) on associating with slice controller $c \in C$ is given as $\Delta_c^f = \max_v(\Delta_{vc}^{prop} y_{vc}^f) + \Delta_c^{proc}(t) g_c^f + \Delta^{net}$. This delay should be less than the delay requirements of the IoT application. Mathematically,

$$\Delta_c^f \le \Delta_{max}^f \quad \forall c \in C, \forall f \in F \tag{3}$$

where the term Δ_{max}^f represents the maximum delay tolerance

[2]The switches are distributed equally among the proxy controllers.
[3]https://www.opennetworking.org/onos/

of the IoT application belonging to flow-space f. Given the decision variables, objectives, and constraints defined above, the flow-space to controller mapping (FCM) problem can be modeled as an integer program (IP), as follows:

$$\begin{aligned} \min_{\mathbf{Y}} \quad & \phi(t) = \sum_{c \in C} \phi_c^{ctrl}(t) + \sum_{c \in C} \phi_c^{serv}(t) \\ \text{subject to} \quad & (1), (2), (3). \end{aligned} \tag{P1}$$

The problem (P1) is non-linear due to $\Delta_c^{proc}(t)$, and is solved using the APOPT[4] MINLP solver. For large instances, the non-linearity in (P1) can be approximated using a piece-wise linear function, and thus solved efficiently using ILP solvers such as Gurobi[5].

IV. Dynamic Scheduling

In this Section, we present a dynamic scheduling scheme to re-compute the flow-space mapping, to better reflect a real-world scenario.

Let $\mathbf{Y}_0$ denote the initial solution of the problem (P1), consisting of the assignment of 0 or 1 to the binary variables $y_{vc}^f(t)$ at time $t = 0$. In a dynamic network environment, the link latencies δ_l can change with time due to factors such as network congestion. The link latency variations, along with the temporal variation in the packet-in rate $\lambda_s^f(t)$ affects Equations (2) and (3), and leads to sub-optimality in $\mathbf{Y}_0$. Thus, the flow-space mapping should be revised periodically by re-computing the integer program (P1). However, the change in mapping is associated with a cost — in terms of communication overhead between the proxy controllers and the slice controllers. Revising the flow-space mapping at each time instance will lead to high overhead, which is not desired. Thus, assuming that the system can tolerate some amount of QoS violations (bounded), we focus on finding the appropriate time to revise the flow-space mapping, so that the expected cost is minimized.

Given two solutions $\mathbf{Y}_0$ and $\mathbf{Y}_t$ with $t > 0$, the cost of re-mapping the flow-spaces is given as $R_{0,t} = \sum_{vcf} \mathbf{1}(y_{vc}^f(0) \neq y_{vc}^f(t))$, where $\mathbf{1}(A)$ is an indicator function which takes value 1 if event A is true, and 0, otherwise. Thus, the expected cost of re-mapping from time 0 to t is given as $E[R_{0,t}] = |V||C||F|P(y_{vc}^f(0) \neq y_{vc}^f(t))$. The temporal variations in link latencies and traffic rate affect the QoS of a flow-space by impacting Equations (2) and (3). Thus, to define the QoS violations of flow-space f at time t, we consider an indicator variable Q_t^f, which takes the value 1 when $(\mathbf{1}(\theta_c > \beta\mu_c) \vee \mathbf{1}(\Delta_c^f > \Delta_{max}^f)) \wedge g_c^f = 1$, and 0 otherwise. In other words, Q_t^f indicates QoS violation of a flow-space f, by checking Equations (2) and (3) and whether the particular slice controller c is assigned to f. Therefore, the total QoS violations at time t is given as $Q_t = \sum_f Q_t^f$. Our aim is to keep track of the development of Q_t with time, and determine the optimal time t^* to re-compute the flow-space mapping in order to minimize the expected cost $E[R_{0,t^*}]$. We utilize the principles of *optimal stopping theory* [17] to find the optimal time t^* for re-computing the flow-space mapping. The cumulative QoS violations from $t = 0$ upto time t is given as $Z_t = \sum_{i=0}^{t} Q_i$. Let Ω denote the tolerance limit of QoS violations. If $Z_t > \Omega$, we should re-compute the flow-space mapping with expected cost $E[R_{0,t}]$. For simplicity, we consider a fixed Ω for all flow-spaces; however, the model may be extended to consider separate tolerance values for different flow-spaces. We want to go as long as possible without revising the flow-space mapping; thus, our aim is to maximize Z_t without violating Ω. Therefore, we define a reward function $\varphi(Z_t)$ such that $\varphi(Z_t) = Z_t$, if $Z_t \leq \Omega$, and $\varphi(Z_t) = -E[R_{0,t}]$, otherwise.

Thus, the problem is to find the optimal time t^* such that $t^* = \sup_{t \geq 0} E[\varphi(Z_t)]$. To solve this problem, we use the one step look ahead (OSLA) rule, which gives a re-computation time $\tau = \inf_{t \geq 0}\{\varphi(Z_t) \geq E[\varphi(Z_{t+1}) \mid \mathcal{F}_t]\}$, where $\mathcal{F}_t$ is the σ-algebra generated by Z_t and denotes the information available until time t, i.e., the sequence $Z_1, \cdots, Z_t$. In other words, the OSLA rule suggests re-computing the flow-space mapping at the first time instance in which the expected reward is as high as continuing to the next instance, and then re-computing. First, we apply the OSLA rule to the given problem, and then show that the time τ given by it is indeed optimal, i.e., $\tau = t^*$.

OSLA rule. Given the information (or *filtration*) $\mathcal{F}_t$, and using the OSLA rule, we re-compute the flow-space mapping at the first time instance which gives $\varphi(Z_t) \geq E[\varphi(Z_{t+1}) \mid \mathcal{F}_t]$ with the event $\{Z_t \leq \Omega\} \in \mathcal{F}_t$. Mathematically, we have

$$\begin{aligned} & E[\varphi(Z_{t+1}) \mid Z_t \leq \Omega] = \\ & E[Z_{t+1} \mid Z_t \leq \Omega, Z_{t+1} \leq \Omega]P(Z_{t+1} \leq \Omega) \\ & + E[Z_{t+1} \mid Z_t \leq \Omega, Z_{t+1} > \Omega]P(Z_{t+1} > \Omega) \text{ [using } \varphi(Z_t)] \\ & = E[Z_t + Q_{t+1} \mid Q_{t+1} \leq \Omega - Z_t]P(Q_{t+1} \leq \Omega - Z_t) \\ & + E[-E[R_{0,t}] \mid Q_{t+1} > \Omega - Z_t]P(Q_{t+1} > \Omega - Z_t) \\ & = \sum_{q=0}^{\Omega - Z_t} (Z_t + q)P_{Q_{t+1}}(q) - E[R_{0,t}]F_{Q_{t+1}}(q) \end{aligned}$$

[since Q_{t+1} is a random variable and independent of Z_t]

where $P_{Q_{t+1}}(q)$ and $F_{Q_{t+1}}(q)$ denote the probability mass function (PMF) and cumulative distribution function (CDF) of the random variable Q_{t+1}. Therefore, according to the OSLA rule, starting from $t = 0$, we revise the flow-space mapping at the first time instance τ such that

$$\begin{aligned} \tau &= \inf_{t \geq 0}\{\varphi(Z_t) \geq E[\varphi(Z_{t+1}) \mid Z_t \leq \Omega]\} \\ &= \inf_{t \geq 0} Z_t \geq \Big[\sum_{q=0}^{\Omega - Z_t} (Z_t + q)P_{Q_{t+1}}(q) - E[R_{0,t}]F_{Q_{t+1}}(q)\Big] \end{aligned} \tag{4}$$

Theorem 1. *The OSLA rule is optimal for monotone stopping problems [17].*

Corollary. *The OSLA rule in Equation* (4) *is optimal for the*

[4]http://apmonitor.com/

[5]https://www.gurobi.com/

FCM problem, i.e., $\tau = t^*$.

Proof. Since Z_t is non-decreasing with t, the difference $E[\varphi(Z_{t+1}) \mid \mathcal{F}_t] - \varphi(Z_t)$ is non-increasing when $Z_t \leq \Omega$. Thus, the stopping problem given as $\{E[\varphi(Z_{t+1}) \mid Z_t \leq \Omega]\}$ is monotone, and hence, by Theorem 1, the OSLA rule for the FCM problem given in Equation (4) is optimal. $\square$

Time Complexity. Equation (4) requires summing over $P_{Q_{t+1}}(q)$ values from 0 to $\Omega - Z_t$. This can be done by recursively summing upto time $t-1$, and from it, subtracting the sum over $E[Z_t - Z_{t-1}]$ terms, which can be at most $|F|$. Thus, the time complexity of the dynamic scheduler is $O(|F|)$.

V. Performance Evaluation

To evaluate the effectiveness of the proposed scheme, we consider two experiments using a python-based simulator. First, we compare the performance of the optimal flow-space to controller mapping scheme (FCM) presented in Section III-B to the fractional controller mapping scheme (SFM) presented in [11]. Second, we analyze the performance of the dynamic scheduler presented in Section IV by comparing it to two alternative strategies. We consider a scale-free Barabasi-Albert model for the data plane topology, and the switches are assigned to the proxy controllers in a uniform random manner. The different simulation parameters are presented Table I.

TABLE I: Simulation parameters

Parameter	Value	Parameter	Value
Switches	20	Proxy controllers	2
Slice controllers	10	Flow-spaces	10
Controller capacity	$100 - 200$	β	0.9

The proposed FCM scheme assigns a distinct slice controller to a particular flow-space, depending on the QoS requirements of the traffic type flowing through that flow-space. On the other hand, the fractional mapping scheme (SFM) attempts to minimize the overall delay by sending a fraction of the flows to different controllers. The controller mapping scheme is dependent on the traffic characteristics; thus, to show the effect of different types of IoT traffic, we consider a mix of real-time, near real-time, and delay-tolerant IoT traffic. The distribution of the mix of traffic types and their delay tolerances are shown in Table II.

TABLE II: Traffic types

Type	Delay (ms)	Dist. 1	Dist. 2	Dist. 3
Real-time	$10 - 15$	0.1	0.3	0.6
Near real-time	$30 - 50$	0.3	0.3	0.3
Delay tolerant	> 100	0.6	0.4	0.1

Figure 3 shows the percentage of QoS violated flows for the FCM and SFM schemes, across the three traffic distributions, while varying the number of flows from 200 to 1000. From the figure, we observe that the proposed FCM scheme outperforms the SFM scheme in all cases. In particular, the relative performance improvement is particularly significant in Distribution 1, where the FCM scheme does not incur any QoS violations upto 600 flows. On average, in Distribution 1, FCM achieves approximately 70% less QoS violations than SFM. The rationale behind this is that in contrast to SFM, the FCM scheme does not attempt to balance the load across the slice controllers. In FCM, the slice controllers are assigned to specific flow-spaces with varying traffic rates, and thus have different processing delays Δ_c^{proc}. Thus, real-time traffic can be mapped to a slice controller with small processing delay. On the other hand, in the SFM scheme, the fractional mapping achieves uniform load balancing across the controllers, which increases the processing delay and violates the QoS of real-time traffic.

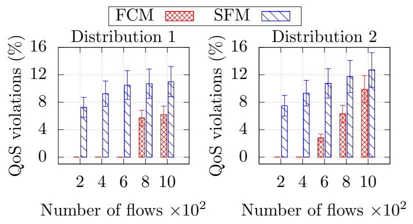

Fig. 3: Comparison of controller mapping schemes

In Distribution 2, having an almost equal mix of 3 types of traffic, we observe that FCM performs worse than in Distribution 1. This is due to the fact that with an increase in real-time traffic, the load on the slice controller assigned to real-time traffic increases, thereby leading to increased processing delay. From the figure, we observe that the proposed FCM scheme performs best with Distribution 1, which reflects the expected traffic composition of IoT, with the majority being delay-tolerant, and few delay-sensitive.

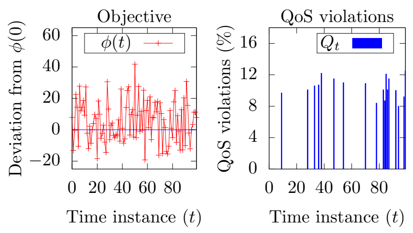

Fig. 4: Variation of objective $\phi(t)$ and QoS violations Q_t with time

Next, we analyze the performance of the proposed flow-space mapping re-computation strategy. Figure 4 shows the variation in the objective function value $\phi(t)$ and the QoS violations Q_t with time. We observe that due to the temporal variations in link latencies and traffic rates, the objective function value given by optimal flow-space assignment (Equation (P1)) deviates significantly from the optimal value at $t = 0$. As shown in the figure, the objective can deviate from -20% to 40% from the optimal objective $\phi(0)$. From the figure, we also observe that temporal variations also lead to QoS violation of $10 - 12\%$ of the flows, due to violating either the delay constraint (Equation (3)) or processing constraint (Equation (2)).

The optimal stopping rule proposed in Equation (4) involves learning the PMF of the distribution of QoS violations in order to calculate the time t^*. Figure 5(a) shows the distribution of the QoS violations Q_t with 1000 flows in the network. The PMF $P_{Q_{t+1}}(q)$ was learnt from the observed data of 1000 time instances using kernel density estimation (KDE). Figure 5(b) shows the performance of three different strategies for flow-space re-computation — the optimal stopping rule strategy proposed in Section IV (top), re-computation at every $t = 50$ time instances (middle), and re-computing once 90% of the QoS violation limit Ω is reached (bottom). From the figure, we observe that the fixed interval strategy at every $50th$ time instance performs the worst, both in terms of number of the re-computations and the number of times it crosses the limit Ω. If the cumulative violation Z_t exceeds Ω in-between 50 time instances, the fixed interval strategy is unable to react and performs poorly. This can be rectified by setting the interval to a low value at the cost of increasing the expected cost $E[R_{0,t}]$. From the figure, it is evident that the optimal stopping rule strategy performs the best, with a 20% (on average) reduction in the number of re-computations required over the simple threshold-based scheme. The proposed scheme adapts to the temporal variation of Q_t, as evidenced by the second re-computation being triggered much later compared to the first. Both the proposed scheme and the threshold-based scheme maintain the QoS limit Ω, however, on average, the proposed scheme achieves an approximate reduction of 34% in the expected cost $E[R_{0,t}]$.

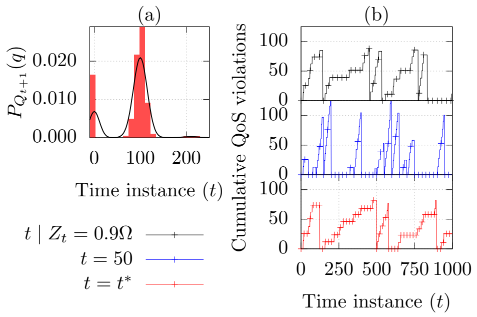

Fig. 5: QoS violations with a) showing distribution of Q_{t+1}, and b) showing cumulative QoS violations Z_t for the three different schedulers

From the analysis above, we see that the proposed flow-space to controller mapping scheme is capable of addressing the QoS requirements of IoT traffic consisting of a mixture of real-time, near real-time, and delay-tolerant flows. Further, the optimal stopping theory based scheduler is capable of adapting to the dynamic variations in traffic and link latencies in IoT networks.

VI. Conclusion

In this paper, we proposed a dynamic network slice assignment scheme for software-defined IoT networks, where each network slice (flow-space) is assigned to a distinct slice controller, based on its QoS requirements. Simulation results show that the proposed optimal flow-space to controller mapping (FCM) scheme is capable of significantly reducing (up to 70%) the QoS violations while considering IoT traffic consisting of real-time, near real-time, and delay-tolerant flows. Further, we proposed a dynamic scheduler, based on optimal stopping theory to take into account the temporal variations in link latencies and traffic, and trigger re-computation of flow-space mapping. Results show that the dynamic scheduling strategy reduces the number and cost of re-computations by 20% and 34% compared to the threshold-based strategy.

Acknowledgments

The authors acknowledge the support received from the University Grants Commission (UGC)-UK India Education Research Initiative (UKIERI) Joint Research Programme (UKIERI-III) under project file No. 184-17/2017(IC).

References

[1] P. Sarigiannidis, T. Lagkas, S. Bibi, A. Ampatzoglou, and P. Bellavista, “Hybrid 5G optical-wireless SDN-based networks, challenges and open issues,” *IET Networks*, vol. 6, no. 6, pp. 141–148, nov 2017.
[2] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, and J. Folgueira, “Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges,” *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 80–87, 2017.
[3] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, “Flowvisor: A Network Virtualization Layer,” OpenFlow Switch Consortium, Tech. Rep., 2009.
[4] E. Salvadori, R. Doriguzzi Corin, A. Broglio, and M. Gerola, “Generalizing Virtual Network Topologies in OpenFlow-Based Networks,” in *Proc. IEEE GLOBECOM*, 2011, pp. 1–6.
[5] N. Saha, S. Bera, and S. Misra, “Sway: Traffic-Aware QoS Routing in Software-Defined IoT,” *IEEE Trans. Emerging Top. Comput.*, 2018, doi: 10.1109/TETC.2018.2847296.
[6] P. Sarigiannidis, T. Zygiridis, A. Sarigiannidis, T. D. Lagkas, M. Obaidat, and N. Kantartzis, “Connectivity and coverage in machine-type communications,” in *Proc. IEEE International Conference on Communications (ICC)*, May 2017.
[7] J. Mocnej, A. Pekar, W. K. G. Seah, and I. Zolotova, “Network Traffic Characteristics of the IoT Application Use Cases,” Victoria University of Wellington, New Zealand, Tech. Rep., 2017.
[8] M. Ojo, D. Adami, and S. Giordano, “A SDN-IoT Architecture with NFV Implementation,” in *Proc. IEEE GLOBECOM Workshops*, Washington, DC USA, Dec. 2016, pp. 1–6.
[9] S. Chaabnia and A. Meddeb, “Slicing aware QoS/QoE in software defined smart home network,” in *Proc. IEEE/IFIP NOMS*, 2018, pp. 1–5.
[10] T. Wang, F. Liu, and H. Xu, “An Efficient Online Algorithm for Dynamic SDN Controller Assignment in Data Center Networks,” *IEEE/ACM Trans. Networking*, vol. 25, no. 5, pp. 2788–2801, 2017.
[11] V. Sridharan, M. Gurusamy, and T. Truong-Huu, “On Multiple Controller Mapping in Software Defined Networks With Resilience Constraints,” *IEEE Commun. Lett.*, vol. 21, no. 8, pp. 1763–1766, 2017.
[12] F. AL-Tam and N. Correia, “Fractional Switch Migration in Multi-controller Software-defined Networking,” *Elsevier Computer Networks*, vol. 157, pp. 1 – 10, 2019.
[13] S. Bera, S. Misra, and N. Saha, “Traffic-aware Dynamic Controller Assignment in SDN,” *IEEE Transactions on Communications*, pp. 1–1, 2020, doi: 10.1109/TCOMM.2020.2983168.
[14] F. Metzger, T. Hoßfeld, A. Bauer, S. Kounev, and P. E. Heegaard, “Modeling of Aggregated IoT Traffic and Its Application to an IoT Cloud,” *Proceedings of the IEEE*, vol. 107, no. 4, pp. 679–694, 2019.
[15] K. He, J. Khalid, S. Das, A. Akella, E. L. Li, and M. Thottan, “Mazu: Taming Latency in Software Defined Networks,” University of Wisconsin-Madison, Tech. Rep., 2014.
[16] Y. Goto, B. Ng, W. K. Seah, and Y. Takahashi, “Queueing analysis of software defined network with realistic OpenFlow–based switch model,” *Computer Networks*, vol. 164, p. 106892, 2019.
[17] T. S. Ferguson, *Optimal Stopping and Applications*. UCLA, 2008. [Online]. Available: https://www.math.ucla.edu/~tom/Stopping/