# Traffic-aware Dynamic Controller Assignment in SDN

Samaresh Bera, *Student Member, IEEE,* Sudip Misra, *Senior Member, IEEE,*
and Niloy Saha, *Student Member, IEEE*

***Abstract*—In this paper, we propose a dynamic controller assignment scheme while considering flow-specific requirements, with an aim to minimize controller response time in software-defined networks (SDN). We adopt the concept of *FlowVisor*, in which a virtualized platform acts as a manager between control- and data planes of SDN architecture. The proposed scheme consists of two phases – adaptive window selection and controller assignment. In the window selection phase, the virtualized manager determines time to wait before incoming flows can be assigned to controllers in adaptive manner. Based on the adaptive window size, the flows are assigned to the controllers in the second phase. We use dynamic stable-matching game to assign flows to controllers, while defining their preference lists. The extensive simulation results show that the proposed scheme is capable of minimizing controller response time by 31%, 39%, and 37% compared to the state-of-the-art schemes – simple stable-matching (SM), static assignment (Static), and minimum quota processing (MQP), respectively. Further, the proposed scheme also reduces the percentage of QoS violated flows in the network by 72%, 73%, and 74% compared to SM, Static, and MQP, respectively.**

***Index Terms*—Software-defined networks, Controller assignment, Stable matching, Adaptive window selection**

## I. Introduction

The advent of software-defined networks (SDN) provides flexible and efficient network management in a centralized manner, while decoupling the control-plane from traditional forwarding devices (such as switches and routers) [1], [2]. Thus, a centralized controller at the control-plane controls the forwarding devices while utilizing the global view of the network. On receiving a flow, i.e., packet, a switch sends a `Packet-In` to the controller, which decides the adequate action(s) to be taken by the switch. The decision making process involves a delay, known as *flow-setup delay*, that depends on the processing capacity of the controller and incoming flow-requests from other switches in the network. To improve scalability and avoid single point failure, cluster of controllers are placed in the network that are statically assigned to switches following master-slave architecture. However, the static assignment yields increased flow-setup delay and control overhead in dynamic network conditions. To address the limitations of static assignment, recently, dynamic controller assignment between switches and controllers is studied [3]. In the dynamic assignment, switches are dynamically assigned to controllers based on their response time.

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

The dynamic assignment between switches and controllers may not be suitable to fulfill the requirements for large number of flow-requests in the network with heterogeneous delay requirements, for example, internet of things (IoT) networks. Figure 1 presents different scenarios of controller assignment and their limitations. It is evident that the existing static and dynamic controller assignment schemes, which considered unspittable flow-spaces, fail in the presence of large number of flow-requests even if the controllers are capable of handling the requests. As shown in Figure 1 (refer to last figure),

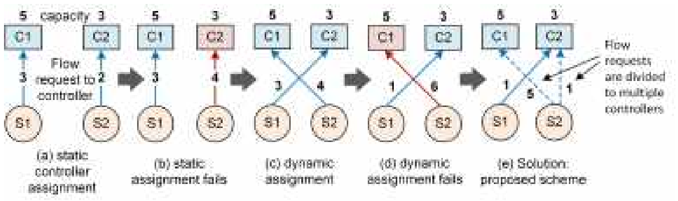

5 capacity 3
C1
C2
Flow request to controller
3
2
S1
S2
(a) static controller assignment
5
3
4
(b) static assignment fails
(c) dynamic assignment
1
6
(d) dynamic assignment fails
5
1
Flow requests are divided to multiple controllers
(e) Solution: proposed scheme

Fig. 1: Motivating scenarios: (a) Static assignment: no problem till flow-requests are within controller capacity; (b) The static assignment fails in the presence of more number of flows; (c) Dynamic assignment resolves the issue by assigning switches to the controller dynamically; (d) Dynamic assignment fails with more number of flows; (e) Proposed approach: flow-requests are dynamically assigned to controllers to resolve the issue.

splitting the flow space among multiple controllers has the potential to address this problem. To this extent, we adopt the *FlowVisor* concept proposed by Sherwood et al. [4], in which a virtualized network platform is used to control flow-requests at a switch using multiple controllers. However, how to dynamically assign flow-requests to controllers remains unaddressed, while considering dynamic network conditions, where traffic fluctuates frequently in an unpredictable manner. Consequently, in this work, we propose a traffic-aware dynamic controller assignment scheme in software-defined networks.

Specifically, we formulate the controller assignment problem as an integer programming problem. As the problem is NP-hard to solve, we propose an online stable matching game-based solution approach to assign flows to controllers. We adopt the concept of a virtualized platform that acts as a manager between control- and data planes, as proposed in [4]. The proposed scheme consists of two phases – adaptive window selection and matching game. In the adaptive window selection phase, the manager adaptively selects window size before assigning flows to controllers. In the matching phase, the concept of dynamic stable matching game is used to assign flows to the controllers according to their preference lists. Extensive simulation results show that the proposed scheme is capable of minimizing controller response time and percentage of QoS violated flows in the network compared to the state-of-the-art schemes – simple stable-matching (SM) [3], static assignment (Static), and minimum quota processing (MQP) [5] schemes. In brief, the contributions in this work are as follows:

- We formulate the controller assignment problem as an integer programming problem, while considering associated constraints.
- As the formulated optimization problem is NP-hard to solve, we propose a dynamic stable-matching game [6] based solution approach to assign flow-requests to controllers dynamically.
- Extensive simulation results show that the proposed scheme is capable of reducing controller response time by 31%, 39%, and 37% compared to the existing schemes – simple stable matching (SM) [3], static assignment (Static) [7], and minimum quota processing (MQP) [5], respectively.

The rest of the work is organized as follows. Section II discusses the state-of-the-art controller assignment schemes in SDN, while highlighting their limitations. We present the system model with preliminaries in Section III. Section IV presents the proposed dynamic controller assignment scheme. The performance of the proposed scheme is presented in Section V. Finally, Section VI concludes the paper with future research directions.

## II. Related Work

In the literature, there exists several schemes that focused on controller-switch assignment in SDN [3], [5], [7]–[15]. We divide the existing schemes in two categories –static and dynamic controller-switch assignment.

### A. Static Assignment

Suh et al. [7] studied the concept of multiple controller assignment problem to minimize flow-setup delay. In such a scenario, on receiving a flow-request from a switch, a controller decides the forwarding path and places flow-rules at the switches associated to the former. Concurrently, the controller informs other controllers associated to the switches that are present in the forwarding path. Thus, the flow-setup delay can be minimized for end-to-end paths. However, in such a scenario, all the controllers need to maintain information about whole network. Moreover, control overhead between controllers increases significantly in the presence of large number of flows in the network. Similar to [7], the controllers need to maintain global network information, which may lead to increased control overhead. Further, the controllers are statically assigned to the switches in the network. Savas et al. [11] proposed a recovery-aware controller-switch assignment technique in SDN. The scheme assigns controllers to switches and determines control-path between them, while planning for a recovery-path. The recovery-path helps to avoid additional control overhead in restoring routing paths in a network failure conditions. The authors formulated the problem as an integer programming problem and proposed heuristic-based solution approach. The authors showed that the proposed scheme is capable of minimizing data-path delay in network failure conditions in the presence of recovery-path, while associating controllers to switches dynamically.

### B. Dynamic Assignment

Wang et al. [3], [8] studied the controller-switch assignment problem from the aspects of simple stable matching. In such a stable matching-based scheme, the matching between switches and controllers is done periodically, which is fixed for all slots. The authors utilized this concept to assign switches based on controllers' response times. Therefore, some of flows are forced to wait for next time-slot before they can be assigned to controllers, which, in turn, leads to QoS-violation of flows in the network. In contrast, the proposed scheme determines the duration of each time-slot dynamically. The duration of time-slot is determined by proposing a dynamic window selection scheme based on online stable matching game proposed by Lee [6]. Thus, the flows are assigned to controllers dynamically depending on their delay requirements (refer to Section IV). Filali et al. [5] proposed a matching game-based assignment technique to associate switches to the controllers, while considering minimum processing quota of controllers. In other words, each controller is assigned to serve a minimum number of tasks in the network. Consequently, a load balancing scheme is ensured in such a scheme. However, the authors did not consider the controller response time while assigning the switches with controllers.

*Synthesis*: Table I summarizes the existing works from different aspects compared to the proposed scheme. The detailed analysis of the existing schemes reveals that they focused on controller-switch assignment without considering flow-specific requirements in the network. This may lead to QoS-violation for some of the flows in the network. To address such issues, we propose a dynamic controller assignment scheme while considering flow-specific requirements.

## III. System Model

Figure 2 represents an SDN-enabled network architecture. We adopt the virtualized controller platform proposed by Sherwood et al. [4]. The virtualized platform helps us to control the controller-switch association dynamically depending on real-time network status. List of symbols used in the work is presented in Table II.

Let the network comprises of a set of switches $\mathcal{S} = \{S_1, S_2, \ldots, S_N\}$, and a set of SDN controllers $\mathcal{C} = \{C_1, C_2, \ldots, C_M\}$. Further, the set of flows in the network is represented as $\mathcal{F} = \{F_1, F_2, \ldots, F_K\}$, where $N, M, K \in \mathbb{Z}^+$. Flow-requests received at the switches are sent to the virtualized platform, and the latter assigns the suitable controller to serve the received request. This is dynamically done in each time-slot. Duration of a time-slot is determined based on

TABLE I: Summary of existing work

| Work | Flow-setup delay | Control overhead | Load balancing | QoS |
|---|---|---|---|---|
| Suh et al. [7], Su and Pack [16] | ✓ | ✗ | ✗ | ✗ |
| Wang et al. [3], [8], Savas et al. [11] | ✓ | ✓ | ✓ | ✗ |
| Filali et al. [5] | ✗ | ✓ | ✓ | ✗ |
| He et al. [9] | ✓ | ✗ | ✓ | ✗ |
| Proposed scheme | ✓ | ✓ | ✓ | ✓ |

TABLE II: List of Symbols

| Symbol | Description |
|---|---|
| $\mathcal{S}$ | Set of switches in the network |
| $\mathcal{C}$ | Set of controllers in the network |
| $\mathcal{F}$ | Set of flows in the network |
| $\alpha_m$ | Request arrival rate at controller $m \in \mathcal{C}$ |
| $\Delta_m^{que}(t)$ | Queuing delay at controller $m$ |
| $\Delta_m^{proc}(t)$ | Processing delay at controller $m$ |
| $\zeta_m$ | Request execution rate of controller $m$ |
| $\theta_m^f(t)$ | Control overhead for flow $f$ at controller $m$ |
| $\Delta_m^f(t)$ | Total delay experienced by flow $f \in \mathcal{F}$ at controller $m$ |
| $\lambda$ | Predefined constant for relative importance on delay and control overhead |

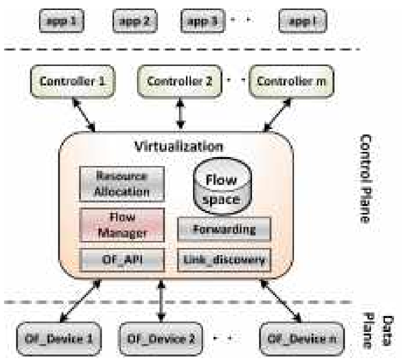


Fig. 2: Software-defined architecture with network resource virtualization

the proposed adaptive window selection algorithm. A flow-request is served by a controller by placing flow-rules at the associated switches in the network. The set of flow-rules associated to a flow-requests is termed as `FlowSpace`, as defined in Definition 1. Consequently, for $K$ flow-requests running in the network, maximum $K$ `FlowSpaces` at a switch can be created. It is noteworthy that there may be many flows crossing difference slices based on the flow-specific QoS requirements[1], as described in FlowVisor. In other words, the flows associated to a particular application cross through the same slice. Whereas flows associated to different applications cross through different slices. We limit our discussion on the number of slices and their architecture in this work as the primary objective is to assign the flows to controllers dynamically while considering their QoS requirements.

[1]In this work, we refer the QoS as the delay-bound, in which the received flow needs to be processed.

**Definition 1.** *`FlowSpace`: A `FlowSpace` is defined as the composition of flow-rules associated with a flow-request. For example, flow-rules associated to a request can be either `exact-match` or `wildcard`.*

## A. Problem Formulation

In this section, we present the flow-setup delay model and control overhead associated to a flow-request while placing flow-rules at the the forwarding devices.

*1) Flow-Setup Delay Model:* The flow-setup delay depends on the propagation delay and controller response time. The propagation delay depends on the path delay to send the request from switch to controller and to place the flow-rules at the switch. Consequently, the propagation delay is represented as follows:

$$\Delta_m^{prop}(t) = \sum_{(i,j)\in P_{s,m}} 2 \times \delta_{i,j} \tag{1}$$

where $P_{s,m}$ denotes the path between switch $s \in \mathcal{S}$ and controller $m \in \mathcal{M}$. The symbol $\delta_{i,j}$ denotes the propagation delay of link $(i,j)$ in the path. On the other hand, the controller response time is the combination of queuing delay and processing delay. The queuing delay depends on the request arrival and execution rates of the controller. At each controller $m \in \mathcal{M}$, flow-requests arrive following unique paths that consists of a set of links in the network. Therefore, according to Kleinrock independence approximation [17], flow-requests arrival rate can be approximated as Poisson process, and it is mathematically denoted as $\alpha_m(t) = \sum_{f\in\mathcal{F}} x_m^f(t)$, where $x_m^f(t)$ is a binary variable to denote whether flow-request $f \in \mathcal{F}$ is associated to controller $m \in \mathcal{C}$ at time $t$. In a multi-threaded scenario, parallel processing of flow-requests at the controllers is considered in this work. Consequently, considering M/M/1 queuing model, the queuing delay experienced by a flow-request at the controller $m \in \mathcal{C}$ is calculated as follows:

$$\Delta_m^{que}(t) = \frac{1}{\zeta_m - \alpha_m(t)} \tag{2}$$

where $\zeta_m$ is the request execution rate of the controller $m$. The request execution rate depends on the forwarding path computation delay by the controller. According to [18], single source forwarding path computation delay depends on the number of switches in the network. Mathematically, the average processing delay experienced by a flow-request in time $t$ at the controller $m \in \mathcal{C}$ is represented as:

$$\Delta_m^{proc}(t) = \frac{1}{\zeta_m} O(|\mathcal{S}|^2) \tag{3}$$

On receiving a new packet, a switch keeps the packet in its buffer and sends a part of the information of the received packet as packet meta-data to the controller. The packet meta-data typically contains packet type with header fields, for example, IPV4 with source and destination address. This packet meta-data is considered as a flow-request to the controllers. The size of the meta-data is the same for all flow-requests, as defined by the OpenFlow protocol [1]. Therefore, the response time of a controller $m \in \mathcal{C}$ for a flow $f \in \mathcal{F}$ is calculated as $\Delta_m^f(t) = \Delta_m^{prop}(t)\Delta_m^{que}(t) + \Delta_m^{proc}(t)$.

*2) Control Overhead Model:* Typically, SDN deployments use in-band communications [8] for both the control messages and data flows in the network. Further, the OpenFlow, the *de-facto* protocol of SDN, allows the switches in the network to send asynchronous control messages to all the controllers in equal state, i.e., master state. In the proposed scenario, the switches send the control messages to the virtualized platform, which takes care of allocating suitable controller. Therefore, each control message is received by at least one controller. The control traffic overhead at time $t$ for flow-request $f \in \mathcal{F}$ associated from switch $n \in \mathcal{S}$ to controller $m \in \mathcal{C}$ is represented as:

$$\theta_m^f(t) = \gamma_{m,n} x_m^f(t), \forall m \in \mathcal{C}, \forall n \in \mathcal{S} \tag{4}$$

where $\gamma_{m,n}$ denotes the hop-count between switch $n \in \mathcal{S}$ and the controller $m \in \mathcal{C}$.

### B. Optimization Problem

The objective of the proposed scheme is to minimize the flow-setup delay and control overhead in the network. Consequently, based on the flow-setup delay and control overhead models, we formulate the optimization problem, while considering associated constraints as follows:

$$\underset{x_m^f}{\text{Minimize}} \;\; \Phi = \sum_{m=1}^{M}\sum_{f=1}^{K} \lambda\Delta_m^f(t) + (1-\lambda)\theta_m^f(t)$$

s. t.

$$\alpha_m(t) \le \zeta_m, \qquad \forall m \in \mathcal{C} \tag{5}$$

$$\sum_{m=1}^{M} x_m^f = 1, \qquad \forall f \in \mathcal{F} \tag{6}$$

$$x_m^f \in \{0,1\}, \qquad \forall m \in \mathcal{C}, \forall f \in \mathcal{F} \tag{7}$$

$$\Delta_m^f(t) \le \delta^f(t), \qquad \forall m \in \mathcal{C}, \forall f \in \mathcal{F} \tag{8}$$

The optimization problem is formulated as a combination of flow-setup delay and control overhead, where $\lambda$ is a predefined constant to consider relative importance on flow setup delay and control overhead. Equation (5) denotes that request assignment rate from FlowVisor to a controller is always less than the processing capacity of the controller. Equation (6) ensures that a flow-request is assigned to only one controller at a time. This preserves the consistency of flow-rules associated to a flow through all switches in the network. The binary variable is used to denote whether a flow is assigned to a controller, as mentioned in Equation (7). Finally, delay-bound of a flow-request is considered Equation (8).

The above mentioned optimization problem is a variant of the generalized optimization problem [19], which is NP-hard. Consequently, we propose an online stable matching-based algorithm to assign the flow-requests to controllers dynamically to minimize associated delay, while considering associated control overhead (refer to Section IV).

### C. Use of online stable matching algorithm

In SDN, controllers take adequate forwarding decisions on receiving a new flow at network devices, i.e., switches and routers. In a single controller scenario, it is straightforward that flow-requests are sent to the controller on receiving new flows. However, in the presence of multiple controllers, it is required to assign flow-requests to the appropriate controller, so that flow-setup delay and associated control overhead are minimized, which is the primary objective of the proposed scheme. Therefore, the objective of the proposed scheme is to find a dynamic mapping between the flow-requests and the distributed controller instances, in order to minimize the controller response time, and thus, meet the heterogeneous delay requirements of flows in the network. These requirements are easily encoded by an optimization model, as presented in Section III-B. However, the problem size can be expected to be relatively large with a large number of traffic flows, which makes it computationally challenging. Since the problem must be solved within a particular delay bound, we explored efficient online heuristics to solve the problem within the stipulated time-bounds.

From the perspective of each flow, the objective is to reduce the flow-setup delay. On the other hand, from a network-wide perspective, the objective is to reduce the overall network overhead. The Gale-Shapely [20] two-sided matching framework lends itself well to adequately model these disparate objectives. In this framework, there are two disjoint sets of agents (i.e., flow-request and controller), each with individual objectives, who rank each other based on their preferences. The problem is to find a suitable matching between these set of agents such that no two agent from each set prefer each other than the current matching. In particular, we consider the online version of the problem, where the agents dynamically enter or leave the system. Online two-sided matching has been studied widely in literature, however, the framework proposed by Lee [6] is particularly suited to model our system, as it considers a delay-bound, within which the matching decisions must be made. This reflects the QoS requirements of the flows in terms of delay. Consequently, we propose a dynamic controller assignment scheme in the presence of multiple controllers and heterogeneous flows in SDN, based on online stable matching approach proposed by Lee [6]. We present the proposed algorithms and discuss key differences between the proposed scheme and the online stable matching strategy proposed by Lee [6] in Section IV.

## IV. Dynamic Controller Assignment

To assign flow-requests to controllers, we define preference lists for flows and controllers and their objectives, similar to the concept proposed by Lee [6]. Therefore, the proposed scheme assigns the flow-requests to controllers dynamically to minimize flow-setup delay, while considering the associated control overhead and flow-specific requirements. It is noteworthy that the proposed algorithms are executed at the FlowVisor platform. Further, introduction of FlowVisor platform between the switches and controllers may add additional flow-setup delay. It is noteworthy that the execution time at the FlowVisor platform is unknown a priori. However, the execution time of the matching algorithm is implicitly considered while evaluating the performance. Therefore, we consider the delay involved in FlowVisor while evaluating the results (refer to Section V for details).

**Objective of flow-request**: Objective of flow-request is to choose the controller with minimum response time. However, the actual response time may vary from the estimated one due to dynamic flow-requests received by controllers in the network. Therefore, a flow-request considers the worst-case response time while choosing a controller to serve it. Mathematically, the worst-case response time is calculated as follows:

$$\Delta_m^{max}(t) = \frac{1}{\zeta_m - \eta_m \zeta_m}, \forall m \in \mathcal{C} \quad (9)$$

where $\eta_m$ denotes the decay factor to handle bursty flow-requests. Therefore, the controller is assigned with maximum load, i.e., flow-requests, which causes to the worst-case response time of a controller $m \in \mathcal{C}$.

**Preference list of flow-request**: According to the worst-case response time mentioned in Equation (9), each flow-request maintains an array of preferences to controllers in ascending order. Mathematically, $P_f(t) = \{C_m\}$, where $m \in [1, M]$, and $I(C_i) \leq I(C_j)$ iff $\Delta_i^{max}(t) \leq \Delta_j^{max}(t), \forall i, j \in [1, M]$ and $i \neq j$. $I(C_i)$ denotes the index of $i^{th}$ controller in the array of flow preference list.

**Objective of controller**: The objective of the controller is to choose flow-requests for which the total cost (combination of delay and control overhead) is minimized (refer to Section III-B). The control overhead associated to a flow-request is calculated as number of hops from the corresponding switch to the controller multiplied by the number of maximum flow-rules associated to the request required to be placed. Mathematically, it is denoted as $\theta_f^{max}(t) = \theta_f m(t) R_f^{max}$, where $\theta_f^{max}(t)$ denotes the maximum control overhead of a controller associated to flow-request $f \in \mathcal{F}$ at time $t$. Symbol $R_f^{max}$ denotes the maximum number of flow-rules associated to the request $f$. Therefore, we redefine the objective function presented in Section III-B as follows:

$$\Phi(f, m, t) = \lambda \Delta_m^f(t) + (1 - \lambda)\theta_f^{max}(t) \quad (10)$$

**Definition 2.** ***$\alpha$% approximated stable matching*** *[6]: A flow-request $f \in \mathcal{F}$ is associated to a controller $m \in \mathcal{C}$, whose rank is within top $\alpha$% of $f$'s original preference list.*

**Definition 3.** ***Matching cost for control overhead*** *(MC): It is defined as the total cost rate of all controllers associated to flow-requests such that $MC = \sum_{m=1}^{M} \Phi(\cdot)$, where $\Phi(\cdot)$ is the cost rate of controller $m$, as presented in Equation (10).*

**Preference list of controller**: According to Equation (10), each controller maintains an array containing preference list of flow-requests in ascending order similar to flow-requests. Mathematically, $P_m(t) = \{F_f\}$, where $f \in [1, F]$, and $I(F_i) \leq I(F_j)$ iff $\Phi(i, m, t)(t) \leq \Phi(j, m, t), \forall i, j \in [1, F]$ and $i \neq j$. $I(F_i)$ denotes the index of $i^{th}$ flow-request in the array of controller preference list.

The flow-requests are assigned to controllers in two phases — adaptive window-size selection and matching between flow-requests and controllers. The phases are discussed in the subsequent sections.

## *A. Adaptive Window-Size Selection*

In this phase, the duration of a time-slot, termed as *window-size*, for taking decisions is determined dynamically depending on the *degree of satisfaction* in matching and associated delay-bound. Therefore, the adaptive window-size selection process tries to maximize the degree of satisfaction, while considering associated delay in order to preserve flow-specific delay-bounds. The degree of satisfaction depends on the approximated stable matching (refer to Definition 2) and the matching cost (refer to Definition 3). Mathematically, the degree of satisfaction $f_{dos}$ is defined as $f_{dos} = \frac{MC}{(\alpha - \frac{1}{M})\tau}$, where $MC$ is the matching cost, $\alpha$ is the approximation ratio of the matching, $M$ is the size of the preference list, i.e., the number of controllers[2]. Symbol $\tau$ denotes the total waiting time of all flow-requests. It is noteworthy that $f_{dos}(MC, \tau) = MC$ if $\alpha - \frac{1}{M}$ or $\tau$ (or both) is zero [6].

According to the value of $f_{dos}$, the virtualized manager determines the duration of window-size in an adaptive manner. Therefore, the window-size changes over time depending on the degree of satisfaction. We limit our discussion on the increasing and decreasing factor in this paper. Interested readers may refer to [6] for details on the window selection algorithm (also presented in supplementary material).

## *B. Matching Algorithm*

In the matching phase, received flow-requests are associated to controllers. Algorithm 1 presents the matching algorithm between flow-requests and controllers, in order to associate flow-requests with controllers.

## *C. Key differences between DCA and [6]*

It is noteworthy that we do not claim to propose a new matching algorithm or improve the performance of an existing one. Rather, our focus is on adopting a suitable approach to efficiently solve the dynamic allocation between flow-requests and distributed controller instances. Thus, we explained the reasoning behind adopting the two-sided matching framework in Section III-C. In particular, the online windowed matching framework proposed by Lee [6] is particularly well-suited to address this problem, since in contrast to other online matching schemes, we do not need to match requests immediately upon arrival, but have some leeway as long as the time remains within the stipulated QoS requirements. However, in contrast to the scheme proposed by Lee [6], we do not consider the

[2]In this work, we consider that the flow-requests initiate the matching process.

**Algorithm 1** Algorithm for matching between flow-requests and controllers

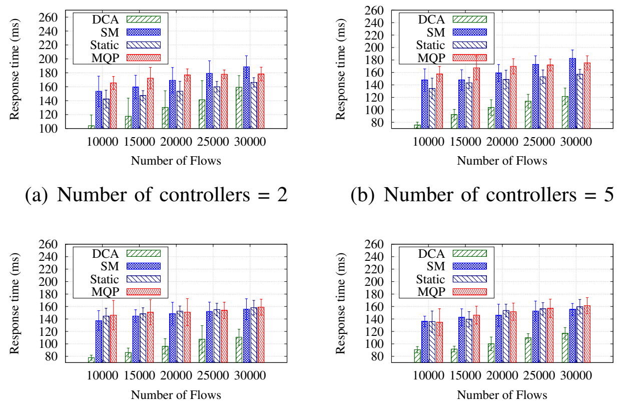
```
Inputs: Preference list of each flow-request $L_f$; Preference list of each controller $L_m$
Output: Stable matching – all flow-requests are assigned
 1: while all requests $f \in \mathcal{F}$ not assigned do
 2:     select controller $m \in \mathcal{C}$ from $L_f$    ▷ controller $m$ is not chosen previously
 3:     if $\alpha_m \leq \zeta_m$ then    ▷ capacity constraint
 4:         assign $f$ to $m$
 5:     else if $\Phi(f, m, t) < \Phi(f^{'}, m, t)$ then    ▷ using equation (10) and preference list $L_m$
 6:         assign request $f$ to controller $m$
 7:         free request $f^{'}$ from controller $m$
 8:     else
 9:         controller $m$ rejects request $f$
10: Stable matching is obtained as all requests are assigned
```

concept of credits. While the concept of credit maximization may be useful in scenarios where each match yields some revenue. In our model, the consideration of credit leads to sub-optimal matching which affects the QoS. Moreover, in contrast to the scheme proposed by Lee, we always consider the flow-requests to be the proposers in the matching game, which leads to a delay-optimal result rather than overhead-optimal. This is significant because the delay has to be within certain given bounds.

## V. Performance Evaluation

We evaluate the performance of the proposed scheme using python-based simulations. Table III shows different parameters and their values used in the simulation [21]. We use the AttMpls network topology from the Internet topology Zoo [22]. The controllers are placed in the network in a uniform-random manner. We plot the results using 95% confidence interval.

We compare the proposed scheme with two existing schemes — simple stable matching (SM) [3], static assignment (Static) [7], and minimum quota of processing (MQP) of controllers [5]. In stable matching (SM), the flow-requests from switches are mapped with controllers according to the preference lists of switches and controllers. The preference list of switches is determined according to the controller response time. On the other hand, switches are statically assigned to controllers in case of static matching (Static). In case of MQP, the switches are assigned to the controllers based on their minimum capacity constraints, i.e., each controller must fulfill their minimum processing quota. Thus, MQP ensures effective load balancing between controllers in the network. In contrast, the proposed scheme, DCA, assigns flows with the controllers according to the preference lists, as presented in Section IV. In rest of the paper, we refer DCA, SM, Static, and MQP to denote the proposed, simple stable matching, static assignment, and minimum processing quota schemes, respectively. Further, we use three performance metrics to show the effectiveness of the proposed scheme — controller



(a) Number of controllers = 2 (b) Number of controllers = 5

(c) Number of controllers = 8 (d) Number of controllers = 10

Fig. 3: Average controller response time with flows

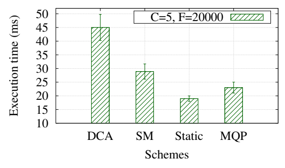
response time, QoS violated flows, and control overhead. We discuss the results in the following subsections.

### A. Controller Response Time

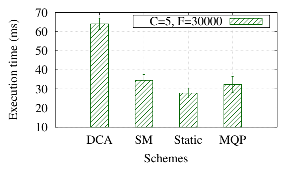
The main objective of the proposed scheme is to minimize flow-setup delay in the network. Consequently, we measure the average controller response time with different number of flows and controllers, as presented in Figure 3. We see that the proposed scheme, DCA, outperforms the existing schemes — SM [3], Static, and MQP [5] — in terms of controller response time. This is due to the fact that the proposed scheme, DCA, assigns flow requests based on their delay requirements and the associated control overheads to the controllers. Therefore, a single controller is not heavily loaded using the proposed scheme, which, in turn, minimizes the overall controller response time. On the other hand, the existing scheme, SM, assigns the flow requests considering delays associated to the controllers only. As a result, it is observed that some of the controllers are heavily loaded, which, in turn, increases the overall controller response time. In case of static assignment, flow requests are statically assigned to controllers irrespective of their response times and associated control overheads. Finally, in case of MQP, flow-requests are equally distributed among the controllers based on their minimum quota of processing. Consequently, the existing schemes, SM, Static, and MQP, yield degraded performance compared to the proposed scheme. Further, we see that controller response time increases with an increase in the number of flows. This is due to the fact that queuing delay at the controllers increases with the increasing number of flows in the network. However, the proposed scheme, DCA, always yields better performance than the existing schemes, SM, Static, and MQP. We also present the average controller response time with different number of controllers in the network, as presented in Figure 3. We see that controller response time decreases with an increase in the number of controllers in the network. This due to the fact that more number of controllers are available to serve flow-requests, which, in turn, minimizes the queuing delay experienced by the flows. It is evident that the proposed

TABLE III: Simulation parameters

| Parameter | Value | Parameter | Value | Parameter | Value |
|---|---|---|---|---|---|
| Number of controllers | [2, 5, 8, 10] | Number of flows | [10000–30000] | Controller exec. rate | 50-100 |
| Average packet size | 94–699 bytes | Network topology | AttMpls [22] | Flow delay bound | 10-200 ms |
| Average traffic rate | 562–516,540 bps | Active time | 1–34 s | Controller response time | 10-100 ms |

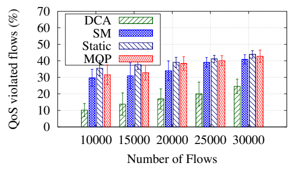


(a) Comparison 1

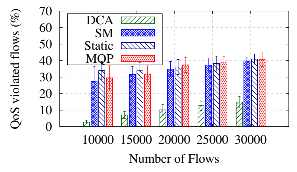


(b) Comparison 2

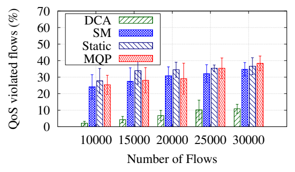
Fig. 4: Comparison of execution time of different algorithms



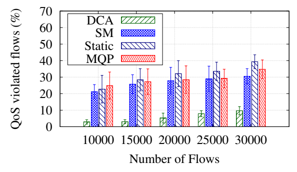
(a) Number of controllers = 2



(b) Number of controllers = 5



(c) Number of controllers = 8



(d) Number of controllers = 10

Fig. 5: QoS violated flows with different number of flows

scheme, DCA, minimizes the controller response time by 31%, 39%, and 37% compared to SM, Static, and MQP, respectively.

It is noteworthy that the proposed scheme, DCA, incurs an additional delay at the virtualized platform in terms of execution time. Figure 4 presents the comparison of execution time between the proposed scheme, DCA, and the existing schemes – SM, Static, and MQP. We see that the execution time is more using the proposed scheme compared to the existing schemes. This is due to the fact that the duration of time-slot and assignment are done dynamically for all flow-requests instead of switches in the network. Therefore, average flow-setup delay is increased using the proposed scheme. However, the proposed scheme is capable of minimizing overall delay compared to the existing schemes, as depicted in Figure 3.

## B. QoS Violated Flows

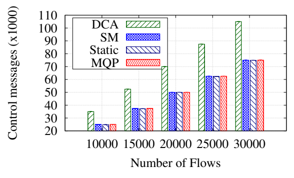
Similar to the controller response time, we also measure the percentage of QoS violated flows in the network, where delay-bound of flow is considered as QoS requirements. Figure 5 shows the percentage of QoS violated of flows with different number of flows. We see that the proposed scheme, DCA, minimizes the QoS violated flows by 72%, 73%, and 74% compared to the existing schemes, SM, Static, and MQP, respectively. The proposed scheme, DCA, assigns the flows to the controllers considering the controller response time and flow delay-bound. Consequently, the percentage of QoS violated flows is minimized. In contrast, the existing schemes, SM and MQP, did not consider the flow delay bounds while assigning flows to the controllers. Moreover, in SM, some of the flows are forced to wait for next time-slot before assignment, which, in turn, leads to increased QoS violation in the network. Similarly, in Static scheme, switches are statically assigned to the controllers irrespective of the flow delay-bounds. Consequently, SM, Static, and MQP yield degraded performances compared to the proposed scheme, DCA, in terms of QoS violated flows. Further, we see that the percentage of QoS violated flows increases with increasing number of flows in the network. This is due to the fact that the controller response time increases with increasing number of flows, as depicted in Figure 3. However, the proposed scheme is always better than the existing schemes. We also see that the percentage of QoS violated flows decreases with increasing number of controllers. This is due to the fact that flow-setup delay is minimized, as controller response time is minimized (refer to Figure 3).

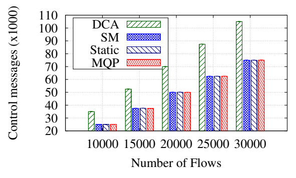
## C. Control Overhead

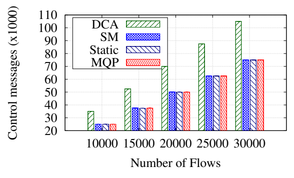
We also measure the control overhead in assigning flow-requests to the controllers including the control messages associated with flow-rule placement, as depicted in Figure 6. The figure shows that the proposed scheme, DCA, incurs increased control overhead compared to the existing schemes, SM, Static, and MQP. This is due to the fact that flow-requests are assigned to the controllers by the virtualized platform. Therefore, each flow-request is received by the virtualized platform, and the latter communicates with all controllers to find the best controller to be assigned. On the other hand, in case of SM, switches communicate with controllers to find the best controller irrespective of flows received by the former. Consequently, SM incurs less control overhead compared to the proposed scheme, DCA, as number of flows are always greater than number of switches in practical scenarios. The similar strategy was considered in case of MQP. On the other hand, switches are statically assigned to the controllers in case of static assignment, which, in turn, incurs less control overhead compared to other schemes.

## D. Impact of Lambda

We show the impact on choosing the value of $\lambda$, which is the relative importance on flow-setup delay and control overhead in the network. Figure 7 depicts the changes in average controller response time and control overhead in the network with different values of $\lambda$. From the results, we see


(a) Number of controllers = 2


(b) Number of controllers = 5


(c) Number of controllers = 8

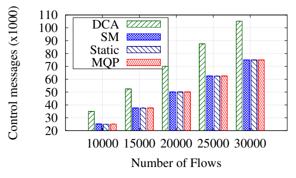
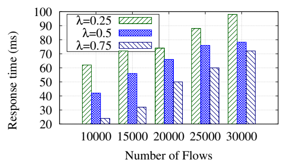

(d) Number of controllers = 10

Fig. 6: Control overhead with different number of flows


(a) Average controller response time

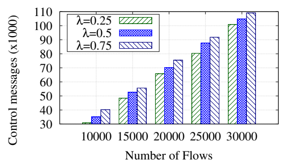

(b) Average control overhead

Fig. 7: Impact of $\lambda$ on controller response time and control overhead

that the value of $\lambda$ can be adjusted based on the user-specific importance on delay and control overhead.

In summary, it is evident that the proposed scheme, DCA, is capable of reducing controller response time and QoS violated flows, while incurring moderately increased control overhead, compared to the existing schemes, SM, Static, and MQP.

## VI. Conclusion

In this paper, we proposed a dynamic controller assignment scheme considering flow-specific requirements. The proposed scheme assigned flows to the controllers based on online stable matching game, in order to minimize flow-setup delay and QoS violation in the network. Extensive simulation results showed that the proposed scheme is capable of enhancing the network performance in terms of delay and QoS violation compared to the existing schemes, while incurring moderate control overhead in the network.

We saw that the proposed scheme incurs increased control overhead in the network due to the inclusion of virtual platform between the data- and control planes. Therefore, we plan to investigate this issue as a future extension of this work. Further, it was observed that the proposed scheme incurs additional execution time due to the inclusion of virtualized platform between switches and controllers. Therefore, we also plan to investigate this in future.

## References

[1] "OpenFlow switch specification," Open Networking Foundation, Tech. Rep., Dec. 2009, version 1.0.0.

[2] S. Bera, S. Misra, and A. V. Vasilakos, "Software-defined networking for internet of things: A survey," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1994–2008, Dec. 2017.

[3] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic SDN controller assignment in data center networks: stable matching with transfers," in *Proc. of the IEEE INFOCOM*, San Francisco, USA, 2016, pp. 1–9.

[4] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "FlowVisor: A network virtualization layer," Deutsche Telekom Inc. R&D Lab, Stanford, Nicira Networks, Tech. Rep., 2009, Accessed on February 12, 2019.

[5] A. Filali, A. Kobbane, M. Elmachkour, and S. Cherkaoui, "SDN controller assignment and load balancing with minimum quota of processing capacity," in *Proc. of the IEEE ICC*, Kansas City, USA, May 2018.

[6] H. Lee, "Online stable matching as a means of allocating distributed resources," *Journal of Systems Architecture*, vol. 45, 1999.

[7] D. Suh, J. Lee, S. Jang, and S. Pack, "Optimal master controller assignment for minimizing flow setup latency in SDN," in *Proc. of the IEEE INFOCOM Workshops*, San Francisco, USA, Apr. 2016.

[8] T. Wang, F. Liu, and H. Xu, "An efficient online algorithm for dynamic SDN controller assignment in data center networks," *IEEE/ACM Trans. on Networking*, vol. 25, no. 5, pp. 2788–2801, Oct. 2017.

[9] M. He, A. Basta, A. Blenk, and W. Kellerer, "Modeling flow setup time for controller placement in SDN: Evaluation for dynamic flows," in *Proc. of the IEEE ICC*, Paris, France, May 2017.

[10] T. Yuan, X. Huang, M. Ma, and J. Yuan, "Balance-based SDN controller placement and assignment with minimum weight matching," in *Proc. of the IEEE ICC*, Kansas City, USA, May 2018.

[11] S. S. Savas, M. Tornatore, F. Dikbiyik, A. Yayimli, C. U. Martel, and B. Mukherjee, "RASCAR: recovery-aware switch-controller assignment and routing in SDN," *IEEE Trans. on Network and Service Management*, vol. 15, no. 4, pp. 1222–1234, Dec. 2018.

[12] N. Correia and F. AL-Tam, "Flow setup aware controller placement in distributed software-defined networking," *IEEE Systems Journal*, 2019, DOI: 10.1109/JSYST.2019.2953771.

[13] M. He, A. Varasteh, and W. Kellerer, "Toward a flexible design of SDN dynamic control plane: An online optimization approach," *IEEE Trans. on Network and Service Management*, vol. 16, no. 4, 2019.

[14] F. Al-Tam and N. Correia, "On load balancing via switch migration in software-defined networking," *IEEE Access*, vol. 7, 2019.

[15] S. S. G. Chalapathi, V. Chamola, C.-K. Tham, S. Gurunarayanan, and N. Ansari, "An optimal delay aware task assignment scheme for wireless SDN networked edge cloudlets," *Future Generation Computer Systems*, vol. 102, pp. 862–875, 2020.

[16] D. Suh and S. Pack, "Low-complexity master controller assignment in distributed SDN controller environments," *IEEE Communications Letters*, vol. 22, no. 3, pp. 490–493, Mar. 2018.

[17] D. P. Bertsekas and R. G. Gallager, *Data networks*, $2^{nd}$ ed. Englewood Cliffs, New Jersey 07632: Prentice Hall, 1992.

[18] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[19] L. Ozbakir, A. Baykasoglu, and P. Tapkan, "Bees algorithm for generalized assignment problem," *Applied Mathematics and Computation*, vol. 215, no. 11, pp. 3782–3795, Feb. 2010.

[20] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *American Math.*, pp. 9–15, 1962.

[21] A. Sivanathan, D. Sherratt, H. H. Gharakheili, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Characterizing and classifying IoT traffic in smart cities and campuses," in *Proc. of the IEEE INFOCOM Workshop*, 2017, pp. 559–564.

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