Distributed Controller-Switch Assignment in 5G Networks

Ehsan Tohidi, Saeedeh Parsaeefard, *Senior Member, IEEE*, Ali Akbar Hemmati, Mohammad Ali Maddah-Ali, *Senior Member, IEEE*, Babak Hossein Khalaj, and Alberto Leon-Garcia, *Life Fellow, IEEE*

***Abstract*—Software defined networking (SDN) is a promising technology in fifth generation wireless networks (5G) where due to the adoption of a centralized SDN-controller, resources such as processing and storage, can be utilized in an optimal manner. Although SDN was first considered with a logically centralized controller, due to delay, reliability, and scalability challenges, moving towards multiple distributed controllers is inevitable. In distributed control schemes, an assignment that associates a controller with each switch leads to three challenges of (1) Computational complexity, since the assignment is an NP-hard problem, (2) Resource and energy efficiency, to obtain an assignment with the lowest number of controllers in order to reduce resource and energy consumption, and (3) Dynamicity, where a dynamic approach of assignment is required to adapt to the network's traffic changes. In this paper, we investigate the controller-switch assignment problem given the aforementioned challenges, and propose efficient algorithms for static and dynamic scenarios, that even achieve quantitative optimality guarantees in special cases. As shown through simulations, the proposed lower complexity algorithms not only outperform earlier works but also approach the performance of exhaustive search schemes, in some scenarios.**

***Index Terms*—Controller-switch assignment, distributed controllers, 5G networks, SDN.**

Manuscript received April 3, 2020; revised November 20, 2020; accepted January 31, 2021. Date of publication March 26, 2021; date of current version June 10, 2021. B. Hossein Khalaj work is supported in part by a grant from the Institute for Research in Fundamental Sciences (IPM). The associate editor coordinating the review of this article and approving it for publication was Y. Li. *(Corresponding author: Ehsan Tohidi.)*

Ehsan Tohidi is with the Department of Telecommunication Systems, Technical University of Berlin, 10623 Berlin, Germany (e-mail: tohidi@tu-berlin.de).

Saeedeh Parsaeefard and Alberto Leon-Garcia are with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada (e-mail: saeideh.fard@utoronto.ca; alberto.leongarcia@utoronto.ca).

Ali Akbar Hemmati and Mohammad Ali Maddah-Ali are with the Department of Electrical Engineering, Sharif University of Technology, Tehran 11365-11155, Iran (e-mail: hemmati.ali.a@gmail.com; maddah_ali@sharif.edu).

Babak Hossein Khalaj is with the Electrical Engineering Department, Sharif University of Technology, Tehran 11365-11155, Iran, and also with the School of Computer Science, Institute for Research in Fundamental Sciences, Tehran 19395-5746, Iran (e-mail: khalaj@sharif.ir).

Digital Object Identifier 10.1109/TNSM.2021.3068979

I. Introduction

A. Motivation

FUTURE networks including fifth generation wireless networks (5G) are anticipated to be identified by their large scale, rapidly changing demands, high availability and flexibility, and wide range of services. Such characteristics necessitate a more flexible and soft network architecture which is based on two main components: software defined networking (SDN) and network function virtualization (NFV). SDN is recognized by its two main features, namely decoupling control plane from data plane and allowing flexible and software-based management of network operations. On the other hand, in NFV, functions are implemented as virtualized network functions (VNFs) on generic high-volume servers. This enables not only fast implementation and agility in response to service requests, but also improves efficiency in resource utilization [1]. In 5G networks, SDN controllers are, in general, assumed to be implemented as VNFs.

One of the tasks of the SDN control plane is to manage the network flows via one centralized SDN controller. Although centralized control in SDN has the advantage of optimal routing for flows, it is not a scalable scheme due to the delay of delivering control messages from the centralized controller to switches, especially in large networks. Moreover, controller failure implies failure of delivering all data flows that depend on the controller. Deploying distributed controllers is a promising scheme to control switches in SDN where there exists a trade-off between performance, delay, reliability, and scalability [2]. However, in this context, an algorithm that associates a controller to each switch is required.

This problem has been studied from different aspects in order to optimize the controller-switch mapping in terms of resource and energy utilization, as the SDN controller consumes a certain amount of resource and energy from the multi-purpose host server, and adapting the assignment to the network condition on a real-time basis, considering the dynamic behavior of the network's traffic, where all face the challenge of high computational complexity particularly for large scale networks where efficient algorithms are required. In order to achieve these goals, one should activate/deactivate controllers (i.e., determining the active set of controllers) based on network conditions, in other words, deactivating under-utilized SDN controllers and distributing the switches'

load among the active ones. We refer to this problem as a controller-switch assignment problem [3].

B. Related Works

The controller selection/placement and controller-switch assignment problems become of high interest recently, e.g., [3]–[8]. For instance, optimizing controller placement to minimize network latency has been proposed in [9]–[14], while the problem of controller placement to maximize network reliability and resilience has been studied in [15]–[19]. Similarly, [20] proposed approaches for controller placement to minimize deployment cost and energy consumption, and [21] formulated the average latency of controlling messages while considering controller failures. Also, in [22], the proposed model considers joint latency constraints, failure tolerance, and load balancing. It should be mentioned that most of the aforementioned works deal with the static assignment, i.e., when network parameters are constant.

Proposing an efficient, practical, and low-complexity algorithm for the aforementioned problem is highly challenging, and to our knowledge has not been addressed properly. In [23], to overcome this challenge, a greedy heuristic algorithm that suboptimally solves the controller-switch assignment problem with the minimum number of active controllers subject to the resource allocation constraints is proposed. Compared to the static problem, dynamic controller placement and assignment has been even less studied. For instance, in [24], a new expansion model for the controller placement problem in SDN is introduced that explores how the network should be re-organized when a set of switches are added to the network. In addition, [22], [23] have proposed online algorithms which perform the tasks of placement and assignment based on heuristic algorithms where neither prior information nor prediction of network traffic is employed. Moreover, a greedy controller-switch assignment algorithm with a given set of selected controllers is proposed in [25] where several controllers can be assigned to each switch in order to achieve load balancing among controllers and consequently, minimizing the maximum processing delay in controllers. In order to protect strategically important controllers from reconnaissance and saturation attacks in SDN, control plane load balancing based on an elastic switch migration model is proposed in [26]. Machine learning based algorithms are recently proposed [27], [28]. Employing deep reinforcement learning, in [27], a dynamic controller placement is proposed which aims to obtain a load balance and minimize the latency. To the best of our knowledge, planned management of controllers in the dynamic scenario (i.e., handling controllers employing prior information) is still an open problem. Consequently, one of our goals is to address the dynamic assignment problem.

Optimization problems of this paper belong to the class of constrained capacitated facility location problems [29]. For instance, a number of papers have addressed the topic of server allocation and consolidation problem [30], [31]. Proving the problem is NP-hard, the proposed solutions either lead to non-polynomial algorithms, such as branch and bound, or require employing heuristic and meta-heuristic algorithms to obtain a suboptimal solution [30]. It should also be noted that as our proposed problem has an extra set of constraints in comparison with existing literature on server allocation and consolidation problem [30], [31], while also being NP-hard, it has not been addressed in earlier works.

C. Contributions and Novelties

In this paper, we aim to address a controller-switch assignment framework, in the direction of 5G networks, where multiple SDN controllers are distributed across the network and are implemented based on NFV. From resource and energy efficiency perspectives, our objective is to find a controller-switch assignment with the minimum number of controllers. In this regard, we consider two scenarios: 1) static scenario where the network parameters are changing slowly and can be considered as constant values, and 2) dynamic scenario where the network parameters change rapidly. We introduce an assignment problem for the static scenario and subsequently, one for the dynamic scenario.

First, the static scenario is modeled and three algorithms with low computational complexity are presented to solve the optimization problem. The pros and cons of the proposed algorithms are discussed and we explain how these algorithms nicely complement each other in different scenarios. Then, we proceed to the more realistic dynamic scenario with time dependent network parameters where we have to adapt the selection and assignment, taking into account the delay for setup time of controllers. We subsequently discuss the interrelation of static and dynamic scenarios, in order to properly extend the proposed static algorithm based on such a relationship. The contributions of this paper can be summarized as follows:

- Considering both static and dynamic scenarios in a unified framework where based on the application and size of the network, and users' dynamic behavior, networks can experience one of these scenarios. Also, properly relating the static and dynamic problems by considering the static problem as the first step and proposing a planning algorithm for activation/deactivation of controllers in dynamic scenarios.
- Improving the model by considering the delay due to the setup time of controllers.
- Presenting three efficient algorithms for the static problem, where the selection among the algorithms is based on a specific network configuration, and providing performance guarantees in special cases.

D. Outline and Notations

The rest of the paper is organized as follows. The system model of controller-switch assignment is introduced in Section II. In Section III, the static scenario is explained and the static controller selection and controller-switch assignment algorithm are proposed. The dynamic scenario is presented and a dynamic selection and assignment algorithm is proposed in Section IV. Section V is dedicated to the numerical results, and finally, Section VI concludes the paper.

We adopt the notation of using boldface lower (upper) case for vectors $\mathbf{a}$ (matrices $\mathbf{A}$). If $\mathbf{A}$ is a 2-dimensional matrix, then $\mathbf{A}_i$ is the ith row of $\mathbf{A}$. Finally, $\{0,1\}^{N\times M}$ is the set of $N \times M$ binary matrices.

II. System Model and Problem Statement

We consider M switches and S multi-purpose servers distributed across the network where each server can host N controllers. The objective is to select the active controllers set, out of the available controllers pool, and also assign a controller to each switch, subject to the network constraints and limitations.

Based on the application and size of the network, and users dynamic behavior, networks can experience smooth or severe changes over time, leading to two scenarios of interest: static and dynamic. In the static case, we consider a network with constant switches' flow, thus, the assignment is applied only once. This case can be considered for the networks with small or smooth traffic changes. In the dynamic case, we have to adaptively change the active controllers set and the controller-switch assignment in order to minimize the objective function (i.e., number of active controllers) and satisfy the constraints (i.e., controllers capacity limitations). On one hand, flows changing over time may cause a controller overflow which requires reassignment or expanding the set of active controllers. On the other hand, it can result in underutilized SDN controllers where via reassignment, we may be able to deactivate some of the controllers.[1]

The problem should be solved for a given time span which we call the assignment period. In the case of dynamic scenarios, we discretize the assignment period into T time slots. Switches' flows can be different from one time slot to another, however, we assume they are constant during a given time slot. For the static scenario, we assume the traffic flow does not have a noticeable variation during one assignment period. In other words, we assume $T = 1$ and drop the time index for static scenarios.

We assume that each switch receives a group of service requests originating from users who are connected to that switch. If the forwarding rule of a request is available in the switch's lookup table, the rule will be followed. Otherwise, the switch should communicate with a controller to determine the optimal routing. The controller-switch communication will naturally require only a part of the request, i.e., the flow header. For each switch, aggregation of the controller-switch communications is called the switch's flow. We denote the value of switches' flow by $\mathbf{F} \in \mathbb{R}^{M\times T}$, where $F_{m,t}$ is flow of the mth switch at the tth time slot. Since controllers are assumed to be implemented as VNFs on multi-purpose servers, where each controller requires a certain amount of resources (i.e., processing and storage) and energy, it is desirable to maintain the assignment by using the minimum number of active controllers. The variable $\mathbf{X} \in \{0,1\}^{S\times N\times T}$ determines the status of the controllers, i.e., $X_{s,n,t}$ is 1 if the nth controller of the sth server is active at the time slot t, otherwise it will be 0.

[1]Due to the possibility of using virtualized SDN controllers (i.e., VNF-based controllers), we do not consider a cost for controllers activation/deactivation.

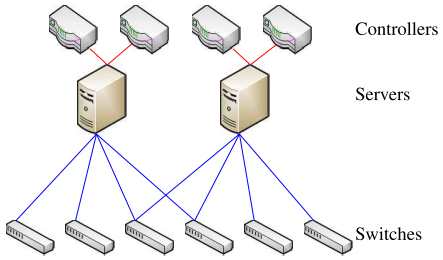


Fig. 1. A server-switch assignability example in a network with 2 servers each with 2 controllers and 6 switches.

The controller-switch assignment is subject to the following two practical constraints:

1) The processing capacity of each controller is limited, i.e., controllers have capacity limitation denoted by $\mathbf{C} \in \mathbb{R}^{S\times N}$ where $C_{s,n}$ represents the capacity of the nth controller of the sth server. Therefore, the sum of flows of the assigned switches to each controller should not exceed its capacity limitation.
2) Only a subset of controllers can be assigned to each switch, leading to the so-called controller-switch assignability restriction. Accordingly, each controller can control only a subset of switches. These restrictions are imposed by two main reasons. First, since both controllers and switches are distributed across the network, there could be some delay constraints due to the large distance among some of the controller-switch pairs. Therefore, such distant controller-switch pairs are not assignable. In addition, a controller-switch pair is assignable if they are following the same service level agreement (SLA). It should be noted that a server can host several controllers and if a switch is assignable to a server, it implies that it is assignable to all other controllers within that server. Consequently, it can be assumed that a switch is, in fact, assignable to a specific server rather than a controller. We model such restrictions by the matrix $\mathbf{R} \in \{0,1\}^{S\times M}$, where $R_{s,m}$ is 1 if controllers of the sth server can control the mth switch, otherwise it will be 0.

The aforementioned constraints are set in order to maintain the required quality of service (QoS) for users (flows). The assignment that satisfies such constraints while minimizing the number of active controllers is denoted by $\mathbf{Y} \in \{0,1\}^{S\times N\times M\times T}$ where $Y_{s,n,m,t}$ is 1 if the mth switch is controlled by the nth controller of the sth server at time slot t, otherwise it will be 0. Figure 1 presents an example of assignability restrictions, showing the three layers of servers, controllers, and switches.

In order to enrich the model, for each controller, we consider a time-delay due to setup time, until it is ready to work. To starting up a virtual machine, the kernel must be loaded into the memory by a bootloader [32]. Based on the size of the kernel, it takes some time to fetch data from a disk to the memory. Also if we want to use containers, some time is needed for the container run time (Docker [33] for instance) to load the necessary parts into the memory [34], [35]. Thus,

this delay is inevitable and must not be ignored to have a more accurate and reliable result out of the simulations. Despite the importance, to the best of our knowledge, this setup time has been never considered. In this paper, we consider a setup time for starting each SDN controller and the dynamic algorithm handles it to conform with real scenarios. Such setup time is of key importance for the dynamic scenario where controller's status alternates between activation and deactivation modes and creates a tendency for the controller to keep its previous status. Consequently, if this issue is not properly addressed, there may be a higher chance to have un-necessary active controllers, as we will elaborate more via simulation results. We assume that the controller setup delay is D time slots. We denote the activation variable by $\mathbf{W} \in \{0,1\}^{S\times N\times T}$ that determines when a controller is in setup, active, or deactivated modes, i.e., $W_{s,n,t}$ is 1 if the nth controller of the sth server is in setup or active mode at time slot t, and 0 if it is deactivated. From the time that one controller is initialized to set up (i.e., even before becoming functionally active), the required resources are reserved and the controller consumes energy. It should be noted that a controller is active at time slot t, if and only if the corresponding entries in $\mathbf{W}$ are set to one for the last D last time slots. We also assume that switching from one controller to another has no cost. This assumption is also based on ETSI-NFV architecture where all information is kept at data repositories of the network orchestrator, and the migration will only be a software update, in addition to small communication messages to announce the new assignment [36].

III. Illustrative Examples

In this section, we highlight different aspects of the controller-switch assignment problem by reviewing some illustrative examples. In the following sections, we will introduce three different algorithms, namely the flow-oriented algorithm (FOA), the controller-oriented algorithm (COA), and the switch-oriented algorithm (SOA). It is interesting to note that, as we will subsequently demonstrate, the algorithm that is suitable for a given problem highly depends on a number of factors, including the network topology. In this respect, the following examples will provide a better intuition on how such factors affect the choice of a proper algorithm. Specifically, we will focus on a fully connected network that matches to FOA conditions, a sparsely connected network which is proper for COA, and also a network topology best suited to SOA.

For illustrative purposes, we will start with the static scenario and a two-layer network including controllers and switches. Figure 2 shows an assignment example of the static scenario where 3 out of 4 controllers are active and a controller is assigned to each switch. As mentioned earlier, we are interested in minimizing the number of active controllers, while all switches are assigned and the assignability and capacity constraints are also satisfied. As will be explained subsequently, this will lead to an NP-hard problem where we look for sub-optimal solutions.

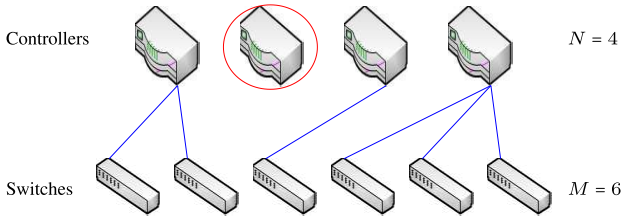


Fig. 2. A sample controller-switch assignment in a network with 4 controllers and 6 switches. All controllers are active except the one encircled in red. The blue lines determine the assigned controller for each switch.

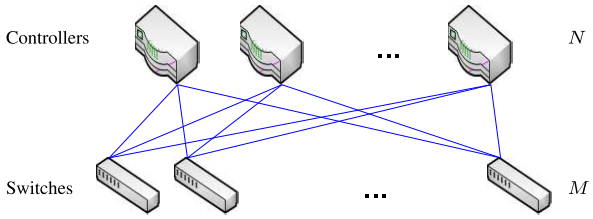


Fig. 3. A fully connected network.

A. First Example - Flow Oriented Algorithm

Here, we consider a fully connected network where all entries of $\mathbf{R}$ are 1 (see Figure 3) and, therefore, each switch can be assigned to any of the controllers. Here, the intuitive approach will be to adopt an approach, which in principle sorts the switches and controllers in descending order, according to their flows and capacities, respectively, and based on that order, each switch is assigned to the lowest indexed controller that has enough capacity. This problem is similar to the classical bin-packing problem [37], where there is a set of objects with different weights and the aim is to fit them in the minimum number of bins, considering an equal capacity for all bins. Correspondence is obtained by substituting switches, flows, and controllers with objects, weights, and bins. There is a famous bin-packing algorithm called First Fit Decreasing (FFD) that solves this problem with 11/9 optimality [38]. The resulting FOA algorithm follows FFD for the controller-switch assignment. Therefore, FOA performs with 11/9 optimality in fully connected networks.

B. Second Example - Controller Oriented Algorithm

Although FOA is intuitively a proper choice in the fully connected controller-switch networks, its performance degrades considerably in the case of sparsely connected networks. For instance, Figure 4 demonstrates an assignment example with $k+1$ controllers and $k+1$ switches where the first switch is connected to the first and second controllers, and for the rest of the switches (i.e., switches 2 to $k+1$), each switch is connected to the first controller and the controller with the same index, i.e., switch m is connected to both the first and the mth controllers for $m = 2,\ldots,k+2$. In this example, we consider capacity 1 for all the controllers, and all the switches

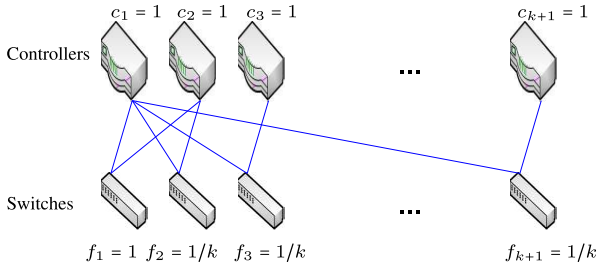


Fig. 4. A sparsely connected network for which COA performs well.

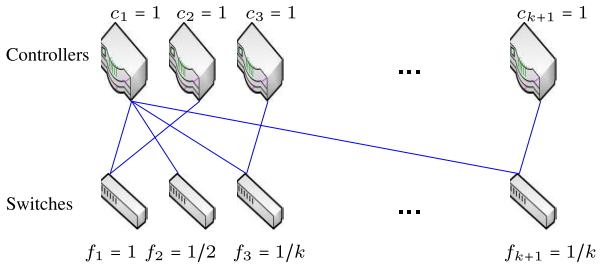


Fig. 5. A sparsely connected network for which SOA performs well.

except the first, have a flow equal to $1/k$. The first switch has a flow of 1.

From Figure 4, the assignment with the minimum number of active controllers is the one that assigns switches 2 to $k+1$ to the first controller and switch 1 to the second controller. Therefore, the optimum assignment requires 2 active controllers. However, employing the FOA, the first switch which has the maximum flow is assigned to the lowest indexed feasible controller which is the first controller. Then, it assigns the remaining switches to the equal indexed controllers. This means that the FOA activates $k+1$ controllers. Clearly, this performance has a large gap with the optimal solution.

The reason for the performance degradation of the FOA in this example lies in the fact that we have to activate k new controllers in order to serve one switch (i.e., the switch with the maximum flow). Such a view suggests that the controller which is able to control the most number of switches (considering both the assignability and capacity restrictions) is the best solution. In this regard, for the example in Figure 4, since the first controller can control up to k switches simultaneously (switches 2 to $k+1$), it should be the first controller to be activated. Consequently, the resulting COA will only require 2 active controllers.

C. Third Example - Switch Oriented Algorithm

Although both FOA and COA seem to be appropriate for a specific set of network topologies, it is also possible to envision that due to the greediness of both algorithms, it would be possible to find scenarios where none of them can find an existing feasible solution. In Figure 5, an example of such an assignment problem is introduced. At first sight, this topology seems to be very similar to the example in Figure 4 except that the second switch is only connected to the first controller and its flow is 1/2. For this topology, it is evident that since the second switch can be controlled only by the first controller, the only feasible assignment for the second switch is to be assigned to the first controller. However, interestingly, neither the FOA nor the COA can find such a solution. In fact, the FOA will assign the first switch to this controller and the COA assigns switches 3 to $k+1$ to the first controller. This observation triggers an idea that in some scenarios, the degree of switches (the number of controllers that can serve a switch) is also an important parameter. In fact, a switch with a lower degree has fewer degrees of freedom; therefore, it might require an additional priority in its assignment. Therefore, for such scenarios, it would be wise to sort the switches in an ascending order based on their degree and perform the switch assignment based on such ordering. For example in Figure 5, the second switch has the lowest degree, and therefore, it should be assigned to the first controller. However, the first controller does not have enough capacity for switch one. Therefore, the first switch will now effectively become a switch of degree one, since the second controller is the only controller that can serve it (considering the capacity constraint). Henceforth, the next candidate switch will now be the first switch, and will now be assigned to the second controller. This procedure will then continue based on the ordering of switches according to their degree, resulting in the SOA. It is also easy to verify that SOA does not provide proper solutions for the first and second examples.

The aforementioned observations lead to the interesting conclusion that to choose a proper assignment algorithm, the network topology and its parameters should be taken into account. An issue that will be more thoroughly discussed in the following section.

IV. Static Switch-Controller Assignment

In this section, we present the static controller selection and controller-switch assignment. Since network parameters are assumed to be constant in this scenario, we drop the time index from all the parameters/variables in this section. In order to distinguish between static and dynamic formulation and prevent ambiguities, for the static case we use the notation in form of $\tilde{\lambda}$, where λ can be any of the variables that are used in both static and dynamic scenarios. The goal is to assign a controller to each switch in such a way that the sum of the flows of the assigned switches to the controller does not exceed its capacity. Simultaneously, we consider restrictions on controller-switch assignability. As mentioned before, each controller is implemented as a VNF which requires a certain amount of resources and energy. Thus, we are interested in a controller-switch assignment that while satisfying the constraints, requires a minimum number of controllers. Controlling parameters for the assignment are $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{Y}}$, where $\tilde{x}_n$ determines whether the nth controller is active (i.e., it is assigned to at least a switch) and $\tilde{Y}_{n,m}$ is a binary variable that determines if the mth switch is assigned to the nth controller.

For this scenario, the optimization problem with the objective to minimize the number of active controllers subject to mentioned practical constraints can be formulated as

$$\begin{aligned} \min_{\tilde{\mathbf{x}},\tilde{\mathbf{Y}}} \ & \sum_{n=1}^{N} \tilde{x}_n \\ \text{s.t.} \ & \text{C1: } \tilde{\mathbf{x}} \in \{0,1\}^{N\times 1}, \tilde{\mathbf{Y}} \in \{0,1\}^{N\times M}, \\ & \text{C2: } \sum_{n=1}^{N} \tilde{Y}_{n,m}\tilde{x}_n = 1, \quad \forall m, \\ & \text{C3: } \sum_{m=1}^{M} \tilde{Y}_{n,m}\tilde{f}_m \le \tilde{c}_n, \quad \forall n, \\ & \text{C4: } \tilde{Y}_{n,m} \le \tilde{R}_{n,m}, \quad \forall n, m, \end{aligned} \tag{1}$$

where C1 states that the optimization parameters are binary, C2 guarantees that each switch is assigned to exactly one controller, C3 enforces controller capacity limitation, and C4 is due to the restrictions on the controller-switch assignability.

This optimization problem, even in its special case with no assignability restriction (i.e., relaxing C4) is well-known to be NP-hard and there is no efficient algorithm (i.e., running in polynomial time) that solves this problem optimally [39]. Although approximation algorithms with proper performance guarantee exist in this literature for the special case (i.e., relaxing C4), the general problem (i.e., the optimization problem in (1)) is rarely studied and we believe that no performance guarantee has been proposed for this problem. Therefore, in the following, we will propose approximation algorithms, which although do not guarantee an optimal solution for every network, provide suboptimal solutions within polynomial time and with a given performance guarantee for some networks.

A. The Proposed Algorithms for Static Assignment

As mentioned in the previous section and through illustrative examples presented there, it has become clear that one single algorithm, even in approximate form, cannot be suitable for all network scenarios. Therefore, in this section, we will go back to the three algorithms mentioned earlier, FOA, COA, and SOA and provide a more formal description and analysis of each of them.

All the three algorithms, i.e., FOA, COA, and SOA, include an initial setup step, followed by an iterative algorithm through which a proper switch or controller (based on the algorithm) is selected. Iterations continue until all switches are assigned taking into account conditions C2-C4. In the following, we provide more details for each algorithm.

1) Flow Oriented Algorithm (FOA): In the first step, FOA sorts the switches in a descending order based on their flows and sorts the controllers in a descending order based on their capacities. Without loss of generality, assume that the switches are sorted as $\tilde{f}_1 \ge \cdots \ge \tilde{f}_M$ and controllers are sorted as $\tilde{c}_1 \ge \cdots \ge \tilde{c}_N$. Starting from the first switch, FOA assigns each switch to the lowest indexed assignable active controller which still has the required capacity. Only when the current switch cannot be accommodated by any active controller, a new assignable controller with the lowest index is activated. The algorithm continues until all the switches are assigned. The proposed algorithm is summarized in Algorithm 1. In the following, we calculate the computational complexity of FOA. The initialization step has a complexity of $O(N\log N + M\log M)$. Through the iterative step, for each switch, we, at most, check all the controllers leading to $O(NM)$ computations. Since the number of controllers is usually less than the number of switches, i.e., $N \le M$, the total computational complexity of FOA is $O(NM + M\log M)$.

Algorithm 1 Flow Oriented Algorithm

Require: $\tilde{\mathbf{R}}$, $\tilde{\mathbf{f}}$.
Ensure: $\tilde{\mathbf{x}}$, $\tilde{\mathbf{Y}}$.
Initialization
$\tilde{\mathbf{x}} \leftarrow \mathbf{0}$, $\tilde{\mathbf{Y}} \leftarrow \mathbf{0}$;
sort $\tilde{\mathbf{f}}$ descending; sort $\tilde{\mathbf{c}}$ descending;
Selection/Assignment procedure
for $m = 1, ..., M$ **do**
  $n^* = 0$
  for $n = 1, ..., N$ **do**
    if $\tilde{R}_{n,m} = 1$ & $\tilde{c}_n \ge \tilde{f}_m$ & $\tilde{x}_n = 1$ **then**
      $\tilde{Y}_{n,m} = 1$, $\tilde{c}_n = \tilde{c}_n - \tilde{f}_m$, $n^* = n$;
      break;
    end if
  end for
  if $n^* = 0$ **then**
    for $n = 1, ..., N$ **do**
      if $\tilde{R}_{n,m} = 1$ & $\tilde{c}_n \ge \tilde{f}_m$ **then**
        $\tilde{Y}_{n,m} = 1$, $\tilde{c}_n = \tilde{c}_n - \tilde{f}_m$, $n^* = n$, $\tilde{x}_n = 1$;
        break;
      end if
    end for
  end if
end for

The FOA near-optimal performance guarantee for fully connected networks is provided according to the following theorem.

Theorem 1: For any fully connected controller-switch assignment problem called *L*, $\mathrm{FOA(L)} < \frac{11}{9}\ \mathrm{OPT(L)} + 4$, where FOA(L) and OPT(L) are the number of active controllers performing FOA and exhaustive search for the controller-switch assignment problem *L*, respectively. From this, it follows that $\lim_{\mathrm{OPT(L)}\to\infty} \frac{\mathrm{FOA(L)}}{\mathrm{OPT(L)}} = \frac{11}{9}$.

Proof: The worst case analysis of FFD for a fully connected assignment has been developed in [40] which is applicable to FOA and proves the theorem. ■

Based on Theorem 1, in scenarios where a controller can control any switch, as might be the case for example, in a typical city-wide network, the FOA performs with 11/9 optimality.

2) Controller Oriented Algorithm (COA): In the initialization phase, we sort the switches in an ascending order based on their flows. Then, based on the assignability restrictions and capacity limitations, we find the maximum number of possible switches (NPS) that can be assigned to each controller. Therefore, for each controller, we start with the first switch and go through the switches one by one. If the switch can be

assigned to the controller, we subtract the switch's flow from the controller capacity. If the remaining capacity is greater than or equal to zero, we increase the NPS by one. Otherwise, the procedure for this controller is terminated. Then, we continue the procedure for the next controller. We denote the NPS of the *n*th controller by γ_n. The possible switches for all controllers are kept in the matrix $\mathbf{P} \in \{0,1\}^{N\times M}$, i.e., $P_{n,m}$ is 1 if the *m*th switch contributes to the NPS of the *n*th controller. In addition, in order to reduce the number of computations, for each controller, we keep the index of the last checked switch which is denoted by the vector $\mathbf{l} \in \mathbb{R}^N$, where l_n determines index of the last checked switch for the *n*th controller. During the assignment step and in each iteration, we select the inactive controller with the maximum NPS, activate it, i.e., $\tilde{x}_{n^*} = 1$ with n^* being the selected controller, and assign the relevant switches (i.e., the switches that contribute to the NPS of the n^*th controller) to this controller, i.e., we change their status into assigned, and adjust the assignment matrix as

$$\forall m, \text{ s.t. } P_{n^*,m} = 1 \rightarrow \tilde{Y}_{n^*,m} = 1, \tilde{z}_m = 1, \qquad (2)$$

where $\tilde{\mathbf{z}} \in \{0,1\}^M$ represents the switches assignment status, i.e., z_m is equal to 1 if the *m*th switch is assigned, otherwise 0. Next, we have to update the NPS for the controllers that any of these switches contribute to their NPS. Subsequently, we add back the flow of that switch to the capacity of the controller (since we reduced it before) and seek among the remaining switches (i.e., for the *n*th controller, continue the search from the l_nth switch) to verify if any of them can be accommodated by that controller, and $\mathbf{l}$ is updated at the end. Based on these steps, the controllers' NPS are updated and we proceed to the next iteration. The proposed algorithm is summarized in Algorithm 2.

In order to calculate the computational complexity of COA, note that sorting the switches based on their flows has the computational complexity of $O(M\log M)$. Initialization of the controllers' NPS is obtained with the total computational complexity of $O(NM)$. For the assignment step, since at least one switch is assigned and one controller is activated at each iteration, there are at most $\min(N, M)$ iterations. In each iteration, we find the controller with the maximum NPS which results in $O(N\min(N, M))$ computations for the controllers selection procedure (i.e., all iterations aggregated). Moreover, the controllers NPS update algorithm has the computational complexity of $O(NM)$ (at most, we check all the switches once for each controller). It should be noted that the computational complexity of the NPS update is given for the whole algorithm (i.e., not per iteration). Therefore, COA has a total computational complexity of $O(NM + M\log M)$.

3) Switch Oriented Algorithm (SOA): In the initializing step of SOA, degrees of switches, i.e., the number of connected controllers to each switch with a capacity greater than or equal to the flow of that switch, are calculated. Therefore, for each switch, and for each controller which is connected to the switch and has enough capacity for flow of the switch, we increase the degree by one and the corresponding entry in the candidate matrix $\mathbf{P} \in \{0,1\}^{N\times M}$ is set to one, i.e., $P_{n,m} = 1$ if the *n*th controller contributes to the degree of the *m*th switch. The degrees vector is represented by $\mathbf{d} \in \mathbb{Z}^M$ where d_m is the degree of the *m*th switch. In the second step, and during each iteration afterward, the unassigned switch with the minimum degree is selected. If there is a currently active controller that is connected to the selected switch and has sufficient capacity, the assignment is done for this switch. Otherwise, a new controller connected to this switch is activated and the controller-switch pair is assigned. After performing the assignment, the switch flow is subtracted from the controller capacity and degrees of switches which are connected to this controller are updated, i.e., for the switches that this controller contributes in their degree, if the updated capacity is less than the switch's flow, its degree is reduced by one. The

Algorithm 2 Controller Oriented Algorithm

Require: $\tilde{\mathbf{R}}$, $\tilde{\mathbf{f}}$.
Ensure: $\tilde{\mathbf{x}}$, $\tilde{\mathbf{Y}}$.
Initialization
$\mathbf{P} \leftarrow \mathbf{0}$, $\tilde{\mathbf{x}} \leftarrow \mathbf{0}$, $\tilde{\mathbf{Y}} \leftarrow \mathbf{0}$, $\tilde{\mathbf{z}} \leftarrow \mathbf{0}$, $\mathbf{l} \leftarrow \mathbf{0}$, sort $\tilde{\mathbf{f}}$ ascending;
for $n = 1, ..., N$ **do**
  for $m = 1, ..., M$ **do**
    if $\tilde{c}_n < \tilde{f}_m$ **then**
      break;
    else if $\tilde{R}_{n,m} = 1$ **then**
      $\gamma_n = \gamma_n + 1$, $\tilde{c}_n = \tilde{c}_n - \tilde{f}_m$, $P_{n,m} = 1$;
    end if
  end for
  $l_n = m$;
end for
Selection/Assignment procedure
while any unassigned switch exists **do**
  $n^* = \arg\max_{\tilde{x}_n=0} \gamma_n$;
  $\tilde{x}_{n^*} = 1$;
  for $m = 1, ..., M$ **do**
    if $P_{n^*,m} = 1$ **then**
      $\tilde{z}_m = 1$, $\tilde{Y}_{n^*,m} = 1$;
    end if
  end for
  Update controllers NPS
  for $m = 1, ..., M$ **do**
    if $P_{n^*,m} = 1$ **then**
      for $n \in \{1, ..., N\} \setminus \{n^*\}$ **do**
        if $P_{n,m} = 1$ **then**
          $P_{n,m} = 0$, $\tilde{c}_n = \tilde{c}_n + \tilde{f}_m$, $\gamma_n = \gamma_n - 1$;
          for $m' = l_n, ..., M$ **do**
            if $\tilde{c}_n < \tilde{f}_{m'}$ **then**
              break;
            else if $\tilde{R}_{n,m'} = 1 \& \tilde{z}_{m'} = 0$ **then**
              $\tilde{c}_n = \tilde{c}_n - \tilde{f}_{m'}$, $\gamma_n = \gamma_n + 1$, $P_{n,m'} = 1$;
            end if
          end for
          $l_n = m'$;
        end if
      end for
    end if
  end for
end while

Algorithm 3 Switch Oriented Algorithm

Require: $\tilde{\mathbf{R}}$, $\tilde{\mathbf{f}}$.
Ensure: $\tilde{\mathbf{x}}$, $\tilde{\mathbf{Y}}$.
Initialization
$\mathbf{P} \leftarrow \mathbf{0}$, $\tilde{\mathbf{x}} \leftarrow \mathbf{0}$, $\tilde{\mathbf{Y}} \leftarrow \mathbf{0}$, $\mathbf{d} \leftarrow \mathbf{0}$, sort $\tilde{\mathbf{f}}$ ascending;
for $m = 1, ..., M$ **do**
  for $n = 1, ..., N$ **do**
    if $\tilde{c}_n \geqslant \tilde{f}_m$ & $\tilde{R}_{n,m} = 1$ **then**
      $d_m = d_m + 1$, $P_{n,m} = 1$;
    end if
  end for
end for
Selection/Assignment procedure
while any unassigned switch exists **do**
  $m^* = \arg\min_{\tilde{z}_m=0} d_m$;
  $\tilde{z}_{m^*} = 1$, $n^* = 0$
  for $n = 1, ..., N$ **do**
    if $P_{n,m^*} = 1$ & $x_n = 1$ **then**
      $\tilde{z}_{m^*} = 1$, $\tilde{Y}_{n,m^*} = 1$, $n^* = n$;
      break;
    end if
  end for
  if $n^* = 0$ **then**
    for $n = 1, ..., N$ **do**
      if $P_{n,m^*} = 1$ **then**
        $z_{m^*} = 1$, $\tilde{Y}_{n,m^*} = 1$, $\tilde{c}_n = \tilde{c}_n - \tilde{f}_{m^*}$, $n^* = n$;
        break;
      end if
    end for
  end if
  Update switches degree
  for $m = 1, ..., M$ **do**
    if $P_{n^*,m} = 1$ & $\tilde{c}_{n^*} < \tilde{f}_m$ **then**
      $P_{n^*,m} = 0$, $d_m = d_m - 1$;
    end if
  end for
end while

proposed algorithm is summarized in Algorithm 3. In terms of computational complexity, the first step has the computational complexity of $O(NM)$. For assigning a controller to each switch, the minimum degree is selected with $O(M)$, the controller is selected with $O(N)$, and the degrees are updated with $O(M)$. Therefore, the total computational complexity of SOA is $O(M^2 + NM)$.

In summary, to solve the static controller-switch assignment problem, the proposed algorithm is to execute the three introduced algorithms and select the best result (i.e., the assignment with the lowest number of active controllers) among them. The total computational complexity will then be $O(NM + M^2)$ and if we assume that the number of switches is greater than the number of controllers, the total computational complexity of the proposed algorithm will be $O(M^2)$ which results in a scalable algorithm even for large scale networks.

V. Dynamic Assignment

We studied the static controller-switch assignment in the previous section. In this section, we proceed to a more complicated model where the network status is time-varying. To model the time-varying characteristic, we discretize the time span into *T* time slots where flow of each switch can be different from one time slot to another. However, we assume this flow remains fixed per time slot. Furthermore, *S* servers are considered all over the network. Controllers are implemented as VNFs in servers and each server can host *N* controllers. Moreover, there are *M* switches and each switch has access to all or none of the controllers of a specific server (based on distance and compatibility). As mentioned earlier, when a switch is assignable to a server, it is assignable to all the VNF-based controllers within that server, i.e., the assignability restrictions are the same for controllers of a server. This result comes from practical considerations that when two controllers are both executed from a given server, they will have equal distances to the switches. In addition, controllers of the same server have similar assignability restrictions. To simplify the illustration, we also assume equal capacity limitation for the controllers of the same server. Note that this assumption does not affect the generality of the algorithm and by sorting the controllers within each server in a descending manner based on their capacities (i.e., controllers with larger capacities having activation priority), the proposed algorithm shall work properly.

Since we implement VNF-based controllers at the servers, there is a delay to set up a controller at a given server. We denote this delay by *D*, which is the number of time slots that are required from the time we decide to activate the controller until it is ready to work. Due to this delay, when the current controllers cannot support the flows of a time slot (e.g., due to increase in flows), we have to set up the required controllers *D* time slots beforehand so that they be ready on time. Therefore, we assume that the flow values of each switch for the next $D + 1$ time slots are known beforehand, and we try to exploit this information to minimize the assignment cost (i.e., the total number of active controllers over time). We also assume that the duration of $D + 1$ time slots is small enough enabling us to obtain such information by either employing traffic prediction algorithms or using probabilistic distribution function (pdf) of traffic obtained through averaging flows over a longer time span.

Remark 1: Assuming the duration of $D + 1$ time slots to be small, employing estimation algorithms, traffic prediction with adequate accuracy is possible.[2] However, in practice and in order to be more confident about the prediction, we can overestimate the traffic, for instance, 10% more than the prediction. Therefore, the controller-switch assignment is valid even for traffic more than the prediction, however, with the price of allocation of extra resources in case of normal traffic. In brief, it is a trade-off between efficiency and reliability.

As mentioned earlier, $\mathbf{W}$ is the activation indicator for the controllers. The dynamic assignment problem is

[2]There are many recent machine-learning based methods, such as the one proposed in [41] that try to predict the traffic in the most efficient manner.

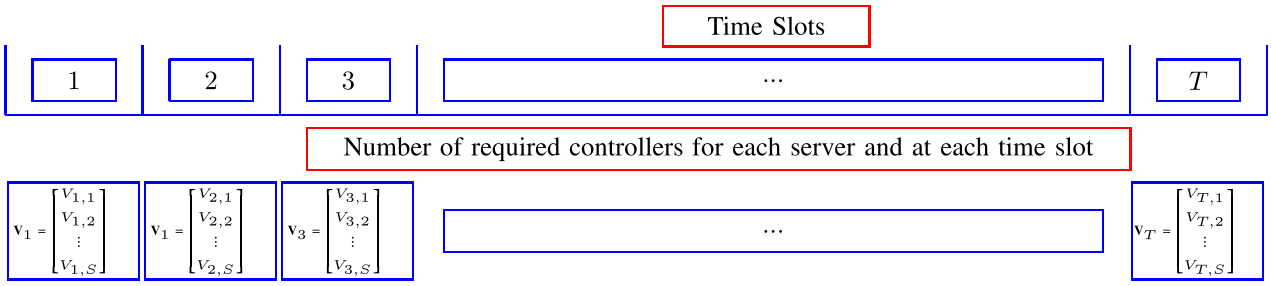


Fig. 6. Solving assignment in time slots as static assignment problems.

formulated as follows:

$$\begin{aligned} \min_{\mathbf{X},\mathbf{Y},\mathbf{W}} \quad & \sum_{t=1-D}^{T} \sum_{s=1}^{S} \sum_{n=1}^{N} W_{s,n,t} \\ \text{s.t.} \quad & \text{C1:} \mathbf{X} \in \{0,1\}^{S\times N\times T}, \mathbf{W} \in \{0,1\}^{S\times N\times (T+D)}, \\ & \text{C2: } \mathbf{Y} \in \{0,1\}^{S\times N\times M\times T}, \\ & \text{C3:} \sum_{s=1}^{S} \sum_{n=1}^{N} Y_{s,n,m,t} X_{s,n,t} = 1, \quad \forall m,t, \\ & \text{C4:} \sum_{m=1}^{M} Y_{s,n,m,t} F_{m,t} \le C_{s,n}, \quad \forall s,n,t, \\ & \text{C5: } Y_{s,n,m,t} \le R_{s,m}, \quad \forall s,n,m,t, \\ & \text{C6: } X_{s,n,t} \le W_{s,n,t-k}, \quad \forall s,n,t,k = 0,\ldots,D, \end{aligned} \tag{3}$$

where the objective function in (3) is the total duration through which the resources are in use. In this case, the objective function will be a function of $\mathbf{W}$ as the resources are reserved and energy is consumed upon setting up a controller. Therefore, even for the time slots that a controller is not active (i.e., the relevant entry in $\mathbf{X}$ is zero), the controller contributes to the cost function if the corresponding entry in $\mathbf{W}$ is equal to one. C1 and C2 state that the optimization parameters are binary, C3 is to ensure that switches are all assigned during the T time slots, C4 is to limit the flows by the available capacity, assignability restrictions are applied in C5, and the delay between setting up a controller until it is ready to use is satisfied by C6 (the relevant entry of $\mathbf{X}$ can be one if corresponding entry in $\mathbf{W}$ is one starting from D previous time slots).

Since the static optimization problem in (1) is a reduced case of (3), the dynamic optimization problem (3) is clearly NP-hard [39] (i.e., setting $D = 0$, $T = 1$, $S = 1$, and $\mathbf{X} = \mathbf{W}$, (3) reduces to (1)). Consequently, there is not an efficient algorithm to solve this problem in an optimal manner. Here, we propose a two-stage heuristic algorithm to solve (3).

A. Proposed Dynamic Assignment Algorithm

Viewing the problem in the time domain, during each time slot (that all the parameters are constant), the optimization problem will reduce to a static problem. This intuition triggers the idea to break the problem into two stages: T static assignments and an integration stage. First, we solve the assignment problem for each time slot as a static assignment. It means that, for each time slot, we have an assignment that determines which controllers are active. Based on these static assignments, the number of required active controllers in each server for each time slot is found. Therefore, it is possible to show the number of required active controllers for each time slot as an S-tuple vectors $\mathbf{V}_t, t = 1,\ldots,T$, where $V_{t,s}$ is the number of required controllers in the sth server at the tth time slot (Figure 6). It should be pointed out that since we assume no difference between controllers running at a specific server, knowing the number of required active controllers of each server is sufficient.

Remark 2: Since only the flow information for the next $D + 1$ time slots are available, and the integration stage (explained later) of the algorithm only needs the number of active controllers during the aforementioned $D + 1$ time slots, the first stage of the algorithm (i.e., running static algorithm for each time slot) is executed for the next $D + 1$ time slots. In other words, when deciding on activation/deactivation of the controllers at the tth time slot, we have only executed the first step for the time slots 1 to $t + D + 1$.

Next, through the integration stage, the controllers can be managed (i.e., activate/deactivate) in such a way that the required number of controllers has become available for each time slot considering the delay D in setting up a controller. The decision on activating/deactivating controllers can be made via an optimal greedy algorithm. Consider $A_{t,s}$ as the number of active controllers at the sth server at the tth time slot. We start the algorithm from time slot $1 - D$.[3] The reason lies beyond the fact that the occurrence of setup procedure takes D time slots. Thus, providing active controllers for the first time slot needs the setup to start at $1 - D$. Clearly, at $1 - D$, we have to set up the controllers exactly based on $\mathbf{V}_1$ and $\mathbf{A}_1 = \mathbf{V}_1$, i.e., the number of required and active controllers are equal for the first time slot. Consequently, we implement $V_{1,s}$ controllers on the sth server at $1 - D$ to be ready for the first time slot.

At time slot $2 - D$-which is related to the 2nd time slot- $\mathbf{A}_1$ controllers have been activated from the previous time slot and $\mathbf{V}_2$ active controllers are required. We split the servers into two groups. The first group contains servers that satisfy

$$A_{1,s} > \max_{t=2,\ldots,D+2} V_{t,s}. \tag{4}$$

These servers have extra controllers that are not required for the next D time slots and thus $A_{1,s} - \max_{t=2,..,D+2} V_{t,s}$ controllers can be deactivated for at least one time slot. For the remaining servers, if $A_{1,s} < V_{2,s}$, $V_{2,s} - A_{1,s}$ controllers are

[3]negative time slots mean that we need to activate some of the controllers before starting the assignment period.

Algorithm 4 The Proposed Algorithm for the Dynamic Controller-Switch Assignment

Require: $\mathbf{R}$, $\mathbf{F}$.
Ensure: $\mathbf{W}$, $\mathbf{Y}$.

Initialization
$\mathbf{A}_0 \leftarrow \mathbf{0}$;
$\mathbf{V}_{t'} \leftarrow \mathbf{0}, \quad t' = T+1, ..., T+D$;

First stage
Solve the static controller-switch assignment for each time slot to obtain $\mathbf{V}$ and $\mathbf{Y}$.

Second stage
for $t = 1, ..., T$ **do**
  for $s = 1, ..., S$ **do**
    $L = \max_{t'=t,..,t+D} V_{t',s}$;
    if $A_{t-1,s} > L$ **then**
      $A_{t,s} = L$;
    else if $A_{t-1,s} < V_{t,s}$ **then**
      for $t' = t-D, ..., t-1$ **do**
        for $n = A_{t-1,s}+1, ..., V_{t,s}$ **do**
          $W_{s,n,t'} = 1$;
        end for
      end for
      $A_{t,s} = V_{t,s}$;
    else
      $A_{t,s} = A_{t-1,s}$;
    end if
    for $n = 1, ..., A_{t,s}$ **do**
      $W_{s,n,t} = 1$;
    end for
  end for
end for

set up at the time slot $2 - D$. As a result, $\mathbf{A}_2$ is calculated as

$$A_{2,s} = \begin{cases} \max_{t=2,...,D+2} V_{t,s}, & \text{if } A_{1,s} > \max_{t=2,...,D+2} V_{t,s}, \\ V_{2,s}, & \text{if } V_{2,s} > A_{1,s}, \\ A_{1,s}, & \text{otherwise.} \end{cases} \tag{5}$$

This procedure is repeated for all the time slots. The proposed algorithm is summarized in Algorithm 4.

In the following, we calculate the computational complexity of the proposed algorithm of dynamic controller-switch assignment for each time slot. We perform the three proposed static controller-switch assignment algorithms with the computational complexity of $O(M^2)$ per each time slot. In the second stage, for each server, we need to compare the required number of controllers for the next *D* time slots. Therefore, the second stage is performed with the computational complexity of *SD*. Therefore, the total computational complexity of the proposed dynamic algorithm per time slot is $O(M^2 + SD)$.

Theorem 2: Given $\mathbf{V}$, the second stage of the proposed algorithm in Algorithm 4 is optimal.

Proof: Based on definition, we have $\forall t, s, A_{t,s} \geq V_{t,s}$. In the proposed algorithm, $A_{t,s} > V_{t,s}$ occurs for a given t and s when there is a $t' \in \{t+1, \ldots, t+D\}$ such that $V_{t',s} > V_{t,s}$. If instead, we deactivate the extra controllers, in order to subsequently compensate the deactivated controllers, we need to set up an equivalent number of controllers from the time slot $\tilde{t} = t' - D$. Clearly $\tilde{t} \leq t$ which means greater or equal cost for setting up new controllers in comparison with the case of keeping the current controllers active. ■

Based on Theorem 2, although determining the required number of active controllers at each server and time slot is not necessarily optimum, the procedure of integration (activating and deactivating the controllers) given these numbers is optimal.

VI. Simulation Results

In this section, we evaluate the proposed algorithms for both the static and dynamic assignments with a different number of switches/controllers and different constraints. We employ an exhaustive search to investigate the gap between the proposed algorithm and the best possible assignment. However, since exhaustive search demands high computational complexity, in the simulations we only consider an exhaustive search for scenarios with a low number of switches and controllers. The closest algorithm to our scheme is the greedy method in [23], here, we also compare it with our algorithms. Although the system model and the proposed algorithms are applicable to the general case of heterogeneous controllers, in the simulations, we consider controllers with equal capacity in order to provide better insight on how the number of required controllers will be different. Throughout the simulations, we normalize all controllers capacity to 1 and switches' flow to the range of [0, 1]. We consider uniform distribution for flows and for a given maximum flow, flows are drawn from the uniform distribution *U*(0, maximum flow). In the following simulations, we consider networks where the number of controllers connected to each switch is equal to the constant value of *Q*. In other words, each switch is connected to *Q* controllers that are selected randomly from a uniform distribution.

A. Static Assignment

In the first scenario, we consider a sample assignment with 20 switches and 10 controllers and a maximum flow of 0.25. Moreover, *Q* is set to 2. Figure 7(a) depicts the assignment restrictions of this sample. Figure 7(b) is the assignment obtained by employing the proposed greedy algorithm leading to 5 active controllers. Figure 7(c) is the result of applying the greedy algorithm in [23] which leads to 7 active controllers at the end. Finally, the assignment using the exhaustive search is presented in Figure 7(d) resulting in 5 active controllers which is equal to the solution of the proposed algorithm. Therefore, for this scenario, the proposed algorithm has obtained the best assignment in terms of the minimum number of active controllers.

For the second scenario, the simulations are implemented for a different number of controllers connected to each switch (i.e., different values of *Q*). We consider 20 switches and 10 controllers. For the first case, the maximum flow is 0.25. Figure 8(a) presents a comparison of the three algorithms.

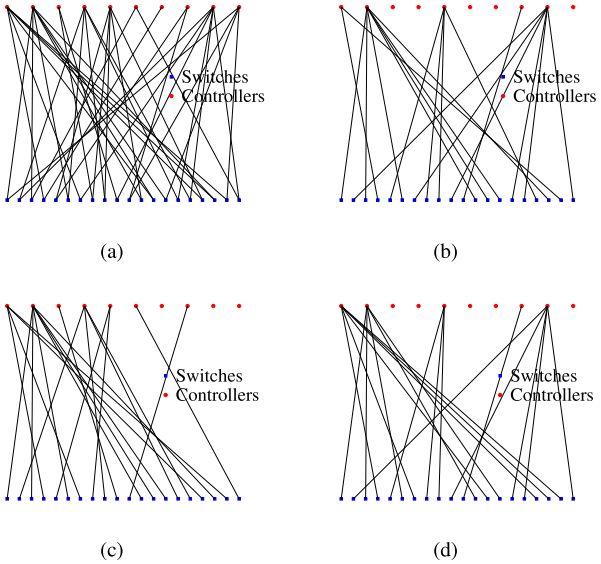


Fig. 7. Controller-switch assignment (a) assignment restrictions, (b) the proposed greedy algorithm, (c) the greedy algorithm in [23], (d) exhaustive search.

Clearly, the number of active controllers decreases as Q is increased (i.e., more degrees of freedom for the assignment). It can be observed that the proposed method outperforms the algorithm in [23] and is approaching the exhaustive search. For instance, for $Q = 3$, the algorithm in [23] requires 40% more active controllers than the exhaustive search while the proposed method is able to identify an assignment with only 8% extra active controllers. Figure 8(b) is a test case similar to 8(a) but with the maximum flow of 0.05. Here, the result of the proposed method approaches to that of the exhaustive search and requires up to 30% lower number of active controllers in comparison with the algorithm in [23]. Via comparing Figures 8(a) and 8(b), it is clear that while the maximum flow is quite different (i.e., one is five times the other), the number of active controllers is nearly equal in both the plots when $Q \leq 5$. However, for $Q > 5$, the number of active controllers in lower flows is less than that for higher flows. These two plots highlight the importance of Q, which corresponds to the role of degrees of freedom in achieving an efficient assignment.

Next, we consider a large scenario with 1000 switches and compare the proposed method and the algorithm in [23] for different parameters. Figures 9(a) and 9(b) depict the number of active controllers versus Q for the maximum flows of 0.25 and 0.05, respectively, where 500 controllers are considered. The plots are obtained by averaging the outcomes of the algorithms with 10 realizations. Again, it is observed that by increasing Q, the number of controllers is reduced. Reducing maximum flow from 0.25 to 0.05 results in reducing the number of active controllers for large values of Q (e.g., $Q = 10$). The superiority of the proposed method is clear from these figures. In fact, for a lower number of connections, the proposed method significantly outperforms the algorithm in [23].

In this case, we study the effect of the maximum flow on the assignment performance in the case of 20 switches. Table I presents the number of active controllers for the three algorithms. Simulations are performed for maximum flows of

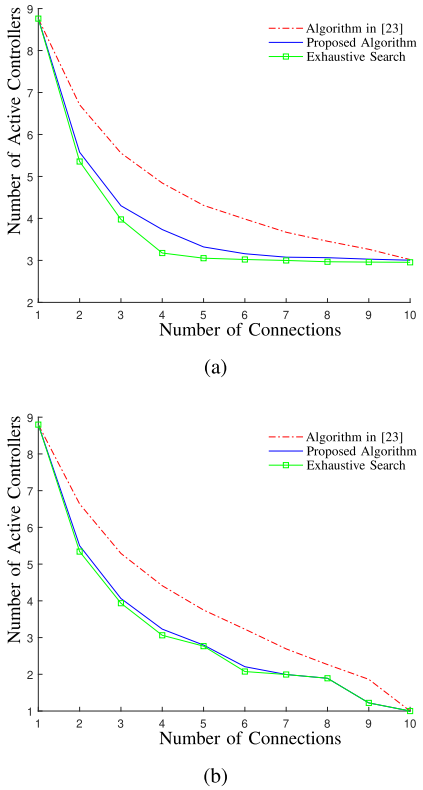


Fig. 8. Number of active controllers versus number of connections Q, for $M = 20$ and max flow of (a) 0.25, (b) 0.05.

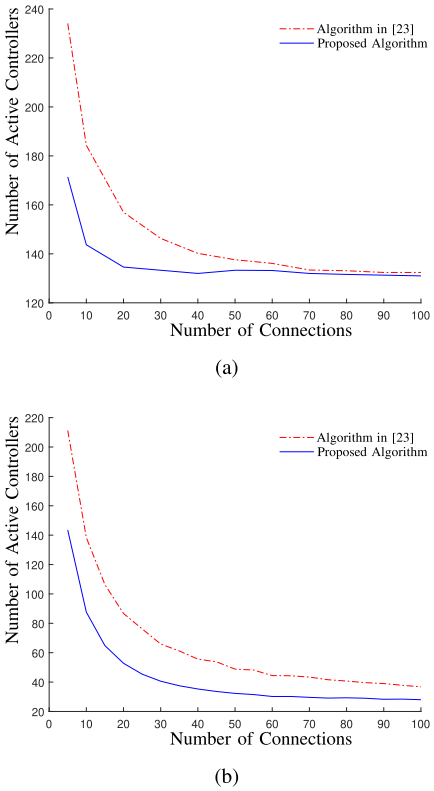


Fig. 9. Number of active controllers versus number of connections Q, for $M = 1000$ and max flow of (a) 0.25, (b) 0.05.

0.05 to 0.5 in the case of a different number of controllers. The results are obtained by averaging over 10 realizations for each algorithm. As expected, increasing the maximum flow leads

TABLE I
COMPARISON OF THE ALGORITHMS' PERFORMANCE VERSUS THE MAXIMUM FLOW FOR $M = 20$

Parameters			Algorithms		
Max flow	Q	N	Alg. [23]	PropAlg	ExhS
0.05	2	2	1	1	1
0.1	3	3	1.5	1.5	1.5
0.15	4	5	2.2	2	2
0.2	4	6	3	2.7	2.4
0.25	4	8	4.2	3.4	3.1
0.3	4	9	5	4	3.4
0.35	4	10	5.7	4.3	4.1
0.4	4	12	6	5	4.3
0.45	4	13	6.7	5.7	5.2
0.5	4	15	7.8	6.3	5.5

TABLE II
COMPARISON OF THE ALGORITHMS' PERFORMANCE VERSUS THE MAXIMUM FLOW FOR $M = 1000$

Parameters			Algorithms	
Max flow	Q	N	Algorithm in [23]	Proposed algorithm
0.005	4	11	7.8	7.2
0.01	6	21	13.1	11
0.02	7	41	24.3	18.4
0.03	8	61	32.5	25.2
0.04	9	81	40.5	30.4
0.05	10	101	46.9	35.1
0.1	12	201	79.6	60.5
0.2	16	401	134.1	110.9
0.3	20	601	183.3	163.7
0.4	24	801	235.9	229
0.5	28	1001	286.4	278.8

to more number of active controllers. The proposed algorithm activates at most 18% more controllers compared to that of the exhaustive search, however, the algorithm in [23] requires up to 47% more active controllers in the worst case.

Table II compares the number of active controllers of the algorithm in [23] and the proposed algorithm with 1000 switches in total. The results are obtained by averaging over 10 network realizations. This case is simulated for different maximum flow while the number of connections and the number of controllers are changing. The proposed algorithm outperforms the algorithm in [23] in all resulting cases.

B. Dynamic Assignment

Here we proceed to evaluate the proposed algorithm in the dynamic scenario. First, we consider a scenario with 10 time slots, delay of 2 time slots, 4 servers each with 10 controllers, and the total number of 400 switches. Each switch is connected to two servers and the value of 0.05 is set for the maximum flow. Figure 10 plots the result of the dynamic assignment for each of the 4 servers. The number of active controllers is often equal to the number of required controllers, except when more controllers are required within a

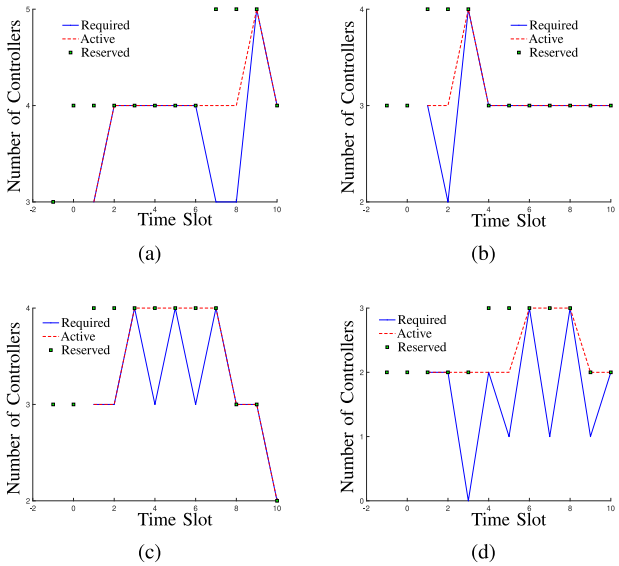


Fig. 10. Number of required, active, and reserved controllers over time for the (a) 1st server, (b) 2nd server, (c) 3rd server, (d) 4th server.

short time (i.e., in the next D time slots). In addition, due to the delay in the controllers' setup time, the number of reserved resources can be more than active controllers, e.g., at 7th time slot in Figure 10(a). The other point to be noted in this simulation is the impact of controllers setup delay on the process of activating/deactivating the controllers. As observed in Figure 10, although the number of required active controllers (i.e., the blue curves) change rapidly during the assignment period, the number of active controllers (i.e., red curves) in servers have smooth change over time, which is due to the existing delay for setup time. Therefore, although controllers activation/deactivation and switches migration are modeled cost-free, the setup delay to some extent reduces the number of rapid alterations in controllers status.

For the second scenario, we compare the proposed algorithm with the exhaustive search. Table III presents the result of simulations for different parameters. For this scenario, we consider four servers each with one controller, $T = 3$, $D = 1$, and $Q = 2$. In this table, we compare the proposed algorithm and the exhaustive search for both the number of active and reserved controllers where the result is averaged over 10 realizations. From Table III, while the number of active controllers is at most 12% more than the exhaustive search, the number of extra reserved controllers can be up to 18%. This is due to the fact that the time slot based arrangement of the controllers is not optimal and as a consequence of setup time delay, the percentile of extra reserved controllers can be higher than the extra active controllers. However, it should be noted that the exhaustive search is only possible for very small scenarios and is not practical in general. Therefore, it is inevitable to accept a tradeoff between optimality and complexity.

In the final case, we study the performance of the proposed algorithm as a function of D in a large scenario where there exist 20 servers each with 4 controllers, 200 switches each with 2 connections, and $T = 20$. Figure 11 depicts the number of active and reserved controllers versus D for different maximum flow values. The result is based on averaging over

TABLE III
COMPARISON OF THE PROPOSED DYNAMIC ALGORITHM WITH EXHAUSTIVE SEARCH VERSUS MAXIMUM FLOW AND M

Parameters		Proposed algorithm		Exhaustive search	
Max flow	M	Active controllers	Reserved controllers	Active controllers	Reserved controllers
0.05	12	6.6	8.8	6.6	8.8
0.1	12	7.8	10.4	7.5	10
0.25	12	8.5	11.5	7.6	10.2
0.25	14	8.8	12.4	8	10.6
0.25	16	9.5	13.3	8.5	11.5

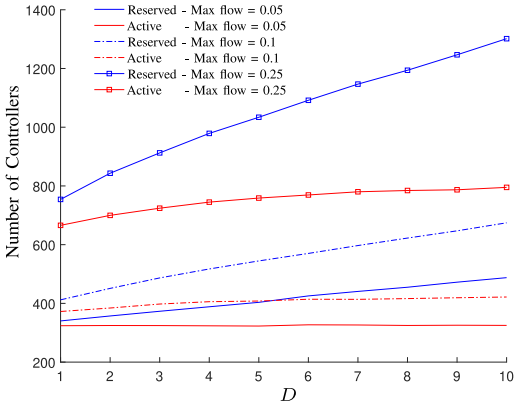


Fig. 11. Performance of the proposed restricted dynamic assignment versus SDN controllers setup delay D for different maximum flow values.

100 realizations. From Figure 11, increasing the maximum flow implies a higher number of controllers. In addition, the number of reserved controllers is proportional to the amount of setup delay since a controller should be reserved from D time slots before becoming active. Although the number of active controllers seems to be independent of the value of delay, it shows a slight increase with increasing delay. The reason stems from the fact that the objective function is to minimize the reserved controllers. Therefore, the optimization algorithm may keep a controller active to prevent setting up a controller in the near future. This can in part lead to a higher number of active controllers.

C. Mininet Implementation

The proposed static algorithms are also verified using Mininet. The codes of this simulation can be found at (github.com/controller-switch-assignment/simulations). To be more realistic, the flow vector is generated using DITG (Distributed Internet Traffic Generator) [42] both in static and dynamic scenarios.

The dynamic algorithm is also applicable in today's cloud-native environments using Kubernetes. Kubernetes or k8s is an open container orchestration tool which was first introduced by Google and now has many contributors around the world [43]. The YAML files of service and deployment in Kubernetes are provided in the GitHub repository. We used this tool to easily scale down and up our controllers which are in a container (called Pod in Kubernetes terminology). Without Kubernetes, it might be more difficult to implement this algorithm, since the controller and switch need some time for handshaking and initial messaging. Consequently, some data are not sent during the initialization process of switch and controller connection. However, using Kubernetes, the switches do not observe what is happening behind the scenes, and thus scaling is performed seamlessly. Kubernetes works on multiple servers but all of them are seen as a single entity from switches' point of view. Therefore, scaling the controllers that are running on servers does not affect the switch side. The proposed dynamic algorithm gives us the number of active controllers at different time slots and using Kubernetes, we scale the number of containers according to that.

In our practical example, we used the OpenDayLight controller. It is a production-level opensource controller and is a Linux Foundation project. "OpenDaylight is a core component of a number of emerging open networking frameworks and integration projects organized around leading use cases" [44]. These frameworks include ONAP and Openstack.

In brief, the flow and capacity constraints are now checked in real scenarios for the proposed algorithms to check their validity and Kubernetes is used to apply the dynamic model.

VII. CONCLUSION

To reach scalability in 5G networks, applying distributed SDN controllers is of particular interest. As a consequence, the controller selection and controller-switch assignment problems can largely affect both network performance and efficiency. In this paper, we studied such assignment problems in both static and dynamic scenarios. In the static scenario where all the parameters are considered to be constant, three algorithms with low computational complexity were presented. Simulation results revealed that the proposed algorithm outperforms the available algorithms in this context. Another key observation of this paper is that the network topology and connections between switches and controllers have a significant impact on the performance of the algorithm used to make such assignment and a single approach cannot be applied to all possible scenarios. In addition, we demonstrated how the problem can be reduced to a bin-packing problem for the special case of a fully connected network, where the proposed algorithm achieves the 11/9 optimality bound. The dynamic selection and assignment algorithm was also presented. It was presented how activation/deactivation of controllers can be managed in order to handle the setup delay of controllers when dealing with a network with dynamic behavior. For future work, we will implement the proposed algorithms over a test-bed using real-world data sets in order to obtain a more realistic evaluation.

REFERENCES

[1] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 518–532, Sep. 2016.

[2] Y. Chang, A. Rezaei, B. Vamanan, J. Hasan, S. Rao, and T. Vijaykumar, "Exploring functional slicing in the design of distributed SDN controllers," in *Proc. Int. Conf. Commun. Syst. Netw.*, 2017, pp. 177–199.

[3] B. Isong, R. R. S. Molose, A. M. Abu-Mahfouz, and N. Dladlu, "Comprehensive review of SDN controller placement strategies," *IEEE Access*, vol. 8, pp. 170070–170092, 2020.

[4] G. Wang, Y. Zhao, J. Huang, and W. Wang, "The controller placement problem in software defined networking: A survey," *IEEE Netw.*, vol. 31, no. 5, pp. 21–27, Sep. 2017.

[5] A. K. Singh, S. Maurya, N. Kumar, and S. Srivastava, "Heuristic approaches for the reliable SDN controller placement problem," *Trans. Emerg. Telecommun. Technol.*, vol. 31, no. 2, 2020, Art. no. e3761.

[6] G. Ramya and R. Manoharan, "Enhanced optimal placements of multi-controllers in SDN," *J. Ambient Intell. Humanized Comput.*, to be published.

[7] E. Tohidi, S. Parsaeefard, M. A. Maddah-Ali, B. H. Khalaj, and A. Leon-Garcia, "Controller-switch assignment in 5G networks," in *Proc. IEEE 2nd 5G World Forum (5GWF)*, 2019, pp. 40–45.

[8] D. Simeonidou, "Joint fronthaul optimization and SDN controller placement in dynamic 5G networks," in *Proc. Opt. Netw. Design Model. 23rd IFIP WG Int. Conf. (ONDM)*, vol. 11616. Athens, Greece, May 2019, p. 181.

[9] Y. D. Lin, P. C. Lin, C. H. Yeh, Y. C. Wang, and Y. C. Lai, "An extended SDN architecture for network function virtualization with a case study on intrusion prevention," *IEEE Netw.*, vol. 29, no. 3, pp. 48–53, May 2015.

[10] J. Liu, Z. Jiang, N. Kato, O. Akashi, and A. Takahara, "Reliability evaluation for NFV deployment of future mobile broadband networks," *IEEE Wireless Commun.*, vol. 23, no. 3, pp. 90–96, Jun. 2016.

[11] G. Wang, Y. Zhao, J. Huang, and Y. Wu, "An effective approach to controller placement in software defined wide area networks," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 1, pp. 344–355, Mar. 2018.

[12] A. Dvir, Y. Haddad, and A. Zilberman, "Wireless controller placement problem," in *Proc. 15th IEEE Annu. Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2018, pp. 1–4.

[13] B. P. R. Killi and S. V. Rao, "On placement of hypervisors and controllers in virtualized software defined network," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 2, pp. 840–853, Jun. 2018.

[14] Y. Li, S. Guan, C. Zhang, and W. Sun, "Parameter optimization model of heuristic algorithms for controller placement problem in large-scale SDN," *IEEE Access*, vol. 8, pp. 151668–151680, 2020.

[15] G. Yao, J. Bi, Y. Li, and L. Guo, "On the capacitated controller placement problem in software defined networks," *IEEE Commun. Lett.*, vol. 18, no. 8, pp. 1339–1342, Aug. 2014.

[16] A. Sallahi and M. St-Hilaire, "Optimal model for the controller placement problem in software defined networks," *IEEE Commun. Lett.*, vol. 19, no. 1, pp. 30–33, Jan. 2015.

[17] M. Tanha, D. Sajjadi, R. Ruby, and J. Pan, "Capacity-aware and delay-guaranteed resilient controller placement for software-defined WANs," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 3, pp. 991–1005, Sep. 2018.

[18] X. Zhang, L. Li, and C. B. Yan, "Robust controller placement based on load balancing in software defined networks," in *Proc. IEEE Int. Conf. Netw. Sens. Control (ICNSC)*, 2020, pp. 1–6.

[19] S. Yang, L. Cui, Z. Chen, and W. Xiao, "An efficient approach to robust SDN controller placement for security," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 3, pp. 1669–1682, Sep. 2020.

[20] Y. Hu, T. Luo, N. C. Beaulieu, and C. Deng, "The energy-aware controller placement problem in software defined networks," *IEEE Commun. Lett.*, vol. 21, no. 4, pp. 741–744, Apr. 2017.

[21] B. P. R. Killi and S. V. Rao, "Capacitated next controller placement in software defined networks," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 3, pp. 514–527, Sep. 2017.

[22] S. Lange *et al.*, "Heuristic approaches to the controller placement problem in large scale SDN networks," *IEEE Trans. Netw. Service Manag.*, vol. 12, no. 1, pp. 4–17, Mar. 2015.

[23] Y. Chen, Y. Yang, X. Zou, Q. Li, and Y. Jiang, "Adaptive distributed software defined networking," *Comput. Commun.*, vol. 102, pp. 120–129, Apr. 2017.

[24] A. Sallahi and M. St-Hilaire, "Expansion model for the controller placement problem in software defined networks," *IEEE Commun. Lett.*, vol. 21, no. 2, pp. 274–277, Feb. 2017.

[25] J. Xie, D. Guo, X. Li, Y. Shen, and X. Jiang, "Cutting long-tail latency of routing response in software defined networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 384–396, Mar. 2018.

[26] Y. Zhou, K. Zheng, W. Ni, and R. P. Liu, "Elastic switch migration for control plane load balancing in SDN," *IEEE Access*, vol. 6, pp. 3909–3919, 2018.

[27] Y. Wu, S. Zhou, Y. Wei, and S. Leng, "Deep reinforcement learning for controller placement in software defined network," in *Proc. IEEE INFOCOM Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2020, pp. 1254–1259.

[28] V. Huang, G. Chen, and Q. Fu, "Effective scheduling function design in SDN through deep reinforcement learning," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2019, pp. 1–7.

[29] A. K. Singh and S. Srivastava, "A survey and classification of controller placement problem in SDN," *Int. J. Netw. Manag.*, vol. 28, no. 3, 2018, Art. no. e2018.

[30] A. Varasteh and M. Goudarzi, "Server consolidation techniques in virtualized data centers: A survey," *IEEE Syst. J.*, vol. 11, no. 2, pp. 772–783, Jun. 2017.

[31] C. Liu, K. Li, and K. Li, "Minimal cost server configuration for meeting time-varying resource demands in cloud centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 11, pp. 2503–2513, Nov. 2018.

[32] A. Silberschatz, P. Galvin, and G. Gagne. (2009). *Control-Based Operating System Design*. [Online]. Available: http://codex.cs.yale.edu/avi/os-book/OS8/os8c/slide-dir/PDF-dir/ch1

[33] D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," *Linux J.*, vol. 2014, no. 239, p. 2, 2014.

[34] *CoreOS Container Linux Startup Process*. Accessed: Mar. 17, 2020. [Online]. Available: https://coreos.com/ignition/docs/latest/boot-process.html

[35] K.-T. Seo, H.-S. Hwang, I.-Y. Moon, O.-Y. Kwon, and B.-J. Kim, "Performance comparison analysis of Linux container and virtual machine for building cloud," *Adv. Sci. Technol. Lett.*, vol. 66, nos. 105–111, p. 2, 2014.

[36] R. Mijumbi, J. Serrat, J. L. Gorricho, S. Latre, M. Charalambides, and D. Lopez, "Management and orchestration challenges in network functions virtualization," *IEEE Commun. Mag.*, vol. 54, no. 1, pp. 98–105, Jan. 2016.

[37] E. G. Coffman, M. R. Garey, and D. S. Johnson, "Approximation algorithms for bin-packing—An updated survey," in *Algorithm Design for Computer System Design*. Vienna, Austria: Springer, 1984, pp. 49–106.

[38] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations* (Wiley-Interscience Series in Discrete Mathematics and Optimization). Hoboken, NJ, USA: Wiley, 1990.

[39] D.-Z. Du and P. M. Pardalos, *Handbook of Combinatorial Optimization*, vol. 1. Norwell, MA, USA: Kluwer, 1998.

[40] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, "Worst-case performance bounds for simple one-dimensional packing algorithms," *SIAM J. Comput.*, vol. 3, no. 4, pp. 299–325, 1974.

[41] C. Zhang, H. Zhang, D. Yuan, and M. Zhang, "Citywide cellular traffic prediction based on densely connected convolutional neural networks," *IEEE Commun. Lett.*, vol. 22, no. 8, pp. 1656–1659, Aug. 2018.

[42] A. Botta, A. Dainotti, and A. Pescapé, "A tool for the generation of realistic network workload for emerging networking scenarios," *Comput. Netw.*, vol. 56, no. 15, pp. 3531–3547, 2012.

[43] J. Baier, *Getting Started With Kubernetes*. Birmingham, U.K.: Packt, 2017.

[44] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDayLight: Towards a model-driven SDN controller architecture," in *Proc. IEEE Int. Symp. World Wireless Mobile Multimedia Netw.*, 2014, pp. 1–6.

Ehsan Tohidi received the B.Sc., M.Sc., and Ph.D. degrees in electrical engineering (with a specialization on communications and signal processing) from the Sharif University of Technology, Tehran, Iran, in 2011, 2013, and 2018, respectively. From September 2016 to March 2017, he was a visiting Ph.D. student with the Department of Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology, The Netherlands. He is currently a Postdoctoral Researcher with the Technical University of Berlin, Berlin, Germany. Prior to that, he was a Postdoctoral Researcher with EURECOM, Biot, France. His research interests include signal processing, discrete and continuous optimization, resource allocation, and machine learning.

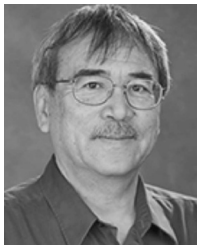
Saeedeh Parsaeefard (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees from the Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran, in 2003 and 2006, respectively, and the Ph.D. degree in electrical and computer engineering from Tarbiat Modares University, Tehran, in 2012. She was a Postdoctoral Research Fellow with the Telecommunication and Signal Processing Laboratory with the Department of Electrical and Computer Engineering, McGill University, Canada. From November 2010 to October 2011, she was a visiting Ph.D. student with the Department of Electrical Engineering, University of California at Los Angeles, Los Angeles, CA, USA. She is currently an Invited Lecturer (Assistant Professor) with the Computer Engineering Department, Amirkabir University of Technology (Tehran Polytechnic) and a Faculty Member with Iran Telecommunication Research Center. Her current research interests include the resource management in software defined networking, Internet of Things, and fifth generation of wireless networks, as well as applications of robust optimization theory and game theory on the resource allocation and management in wireless networks.

Ali Akbar Hemmati received the B.Sc. degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 2019. He is currently pursuing the M.Sc. degree with the Electrical Engineering Department, Sharif University of Technology. His research interests include cloud, cloud native applications, cloud orchestration, and containers.

Mohammad Ali Maddah-Ali (Senior Member, IEEE) received the B.Sc. degree in electrical engineering from the Isfahan University of Technology, the M.A.Sc. degree in electrical engineering from the University of Tehran, and the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Waterloo, Canada, in 2007.

From 2007 to 2008, he was with the Wireless Technology Laboratories, Nortel Networks, Ottawa, ON, Canada. From 2008 to 2010, he was a Postdoctoral Fellow with the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley. Then, he joined Nokia Bell Labs, Holmdel, NJ, USA, as a Communication Network Research Scientist. He is currently working with the Sharif University of Technology as a Faculty Member. He is a recipient of the NSERC Postdoctoral Fellowship in 2007, the IEEE International Conference on Communications Best Paper Award in 2014, the IEEE Communications Society, the IEEE Information Theory Society Joint Paper Award in 2015, and the IEEE Information Theory Society Paper Award in 2016. He is currently serving as an Associate Editor for the IEEE TRANSACTIONS ON INFORMATION THEORY.

Babak Hossein Khalaj received the B.Sc. degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 1989, and the M.Sc. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, USA, in 1993 and 1996, respectively. He has been with the Pioneering Team, Stanford University, where he was involved in adoption of multiantenna arrays in mobile networks. He joined KLA-Tencor in 1995, as a Senior Algorithm Designer, working on advanced processing techniques for signal estimation. From 1996 to 1999, he was with Advanced Fiber Communications and Ikanos Communications. Since then, he has been a Senior Consultant in the area of data communications, and a Visiting Professor with CEIT, San Sebastian, Spain, from 2006 to 2007. He has coauthored many papers in signal processing and digital communications and holds four U.S. patents. He was a recipient of Alexander von Humboldt Fellowship from 2007 to 2008, and the Nokia Visiting Professor Scholarship in 2018. He was also the Co-Editor of the Special Compatibility Standard Draft for ANSIT1E1 Group from 1998 to 1999.



Alberto Leon-Garcia (Life Fellow, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from the University of Southern California, Los Angeles, CA, USA, in 1973, 1974, and 1976, respectively.

He was a Founder and a CTO of AcceLight Networks, Ottawa, ON, Canada, from 1999 to 2002, which developed an all-optical fabric multiterabit, multiservice core switch. He is currently a Professor of the Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada. He holds a Canada Research Chair in Autonomic Service Architecture. He holds several patents and has published extensively in the areas of switch architecture and traffic management. His research team is currently developing a network testbed that will enable at-scale experimentation in new network protocols and distributed applications. He is recognized as an Innovator in networking education. In 1986, he led the development of the University of Toronto-Northern Telecom Network Engineering Program. He has also led in 1997 the development of the Master of Engineering in Telecommunications program, and the communications and networking options in the undergraduate computer engineering program. He has authored of the leading textbooks *Probability and Random Processes for Electrical Engineering and Communication Networks: Fundamental Concepts and Key Architecture*. His current research interests include application-oriented networking and autonomic resources management with a focus on enabling pervasive smart infrastructure. He received the 2006 Thomas Eadie Medal from the Royal Society of Canada and the 2010 IEEE Canada A. G. L. McNaughton Gold Medal for his contributions to the area of communications. He is a Fellow of the Engineering Institute of Canada.