SDN Controller Placement at the Edge: Optimizing Delay and Overheads

***Abstract*—Nowadays, *edge*, also termed *fog*, architectures are becoming a popular research trend to provide elastic resources and services to end-users at the extremes of the network, as opposed to traditional data-centers. Despite their momentum, the control plane of these architectures remains complex and challenging to implement. To enhance control capability, in this work, we propose to use Software Defined Networking. SDN moves the control logic off data plane devices and onto external network entities, the controllers. We provide a proof-of-concept implementation of a multi-controller edge system and measure traffic delay and overheads. The results reveal the sensitivity of delay to the location of controllers and the magnitude of inter-controller and controller-node overheads. Guided by the above, we model the problem of determining the placement of controllers in the edge network. Using linearization and supermodular function techniques, we present approximation solutions which perform close to optimal and better than state-of-the-art methods.**

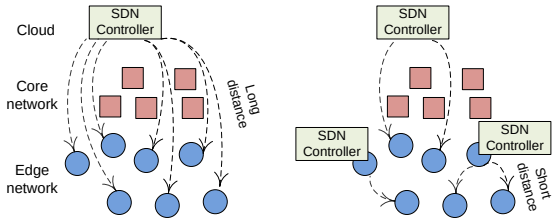
I. Introduction

A. Motivation

Nowadays, there is an increasing interest in novel network architectures which distribute substantial amounts of data storage, processing and communication resources at the extremes of the network [1]. These architectures, often referred to as *edge* or *fog*, target to solve inherent scalability and performance issues of traditional data center and cloud systems by enabling the processing and sharing of data locally at the edge nodes instead of central servers. Although still at an abstract level, edge architectures are considered to be a key enabler for next-generation wireless (5G) [2] and Internet of Things (IoT) [3] systems promising to deliver a rich bouquet of new services to end-users.

However, resource management in edge architectures is a very complex task, especially when a diverse set of services with different requirements need to be supported [4]. Software Defined Networking (SDN) is a new technology, popular in data center and wired ISP networks [5], which can be potentially applied to the edge part of the network to effectively provision and orchestrate its resources. The main principle of SDN is to shift all the control functions from the data plane nodes to a programmable network entity, the controller. However, *this is centralized approach, while edge architectures give emphasis on the distribution of resources and their management*. Therefore, it is challenging to apply SDN ideas at the edge part of the network.

To exemplify, in order for SDN protocol to work properly, the state of the data plane nodes[1] should be reported to the centralized controller in a *timely manner* to make efficient resource management decisions. This condition is easier to be met in wired networks where the communication between the controller and the nodes is much more stable and faster than in the wireless counterpart. In the former case, one controller located at the cloud may suffice to manage in time all the nodes despite the long distance between them (Figure 1(a)). In edge architectures with unstable wireless links, however, the controller located at the cloud may be the bottleneck and fail to maintain an acceptable level of performance.



(a) Single controller. (b) Distribution of controllers.

Fig. 1. A remote controller vs many controllers placed at the edge.

A method that can be used to solve the performance bottleneck of the remote controller is the *placement of many controller instances in proximity to the edge nodes*, as it is depicted in Figure 1(b). The placed controllers, physically distributed but logically centralized, cooperate to manage the edge nodes which can reduce delay due to the shorter distance to them. Such placement strategies are possible today via commercial software implementations of controllers that can be installed in large quantities and operate even in lightweight network devices [7], [8].

The controller placement problem has been extensively studied over the past five years, especially in the context of data center and wired ISP networks (e.g., see the pioneer work in [9] and follow-up works). However, this problem obtains an interesting new twist in the context of edge architectures for the following reasons:

1) *Resource-constrained edge nodes*: Certain edge nodes (e.g., IoT devices) may not be powerful enough to host an SDN controller or the operation of SDN may quickly drain their resources (e.g., processing power). Hence, there is a need to exclude these nodes when deciding the controller placement.
2) *Delay of network management*: Certain links between the edge nodes may be wireless in nature, unstable and of low rate. Moreover, it may happen that many of these links separate a node from a placed controller resulting

[1]The state of the nodes may include traffic statistics, link metrics and other protocol-specific parameters [6].

in slow statistic collection and node re-configuration. Hence, the controller placement strategy can drastically affect the delay of network management.

3) *Overhead of control messages*: Multi-controller implementations require the periodic exchange of messages between the controllers and nodes for statistic collection [6] as well as between the controllers themselves for synchronization purposes [10]. The overheads of these two types typically increase with the number of placed controllers, their distance to the nodes and to each other. Hence, if the controllers are not properly placed, the overheads wil be significant, considering the scarcity of edge network resources.

Given the above issues, the key open questions are: *How many controllers to place in the network and where exactly? Should we place many controllers close to the edge nodes to reduce delay of resource management or place fewer controllers close to each other to keep synchronization overheads as low as possible?*

B. Methodology and Contributions

In this paper, we follow a systematic methodology in order to answer the above questions. We begin by showing the *feasibility of an edge system with multiple SDN controllers* through a testbed built from off-the-shelf edge network devices. The latter are programmed to run Open VSwitch data plane [11] and ONOS controller [8] implementations. We perform experiments to measure the delay of managing a data plane device by a controller device and show its sensitivity on the controller placement strategy. In order to analyze large-scale systems consisting of hundreds of devices (which goes beyond our testbed), we provide an emulation study based on the same controller implementation. We find that inter-controller and controller-node traffic overheads can be at the same order of magnitude (up to a few Mbps each). Moreover, we show that the inter-controller traffic closely resembles a linear function to the controller load with a constant term.

We build upon our experimentation and emulation findings to model the controller placement problem in an edge network. We formulate this problem for the objectives of delay and overhead minimization. As an initial attempt to solve our problem, we convert it to a Mixed Integer Programming (MIP) problem using linearization techniques [12]. This is important since there exist various commercial online solvers, such as CPLEX, which can be directly used to solve this type of problems. Going one step further, we present solution algorithms that scale well with the size of the problem instance. We prove the approximation guarantees of these algorithms using the theory of supermodular functions. Finally, we perform evaluations to show the merits of the proposed algorithms.

The contributions of this work can be summarized as follows:

- *SDN Controller Placement at the Edge*. We propose and study the placement of controllers in SDN-enabled edge networks. We consider several practical features of these systems such as the limited resources of the edge nodes, the different delay values of the wireless links and the impact of control overheads.
- *Experimental Analysis - Delay*. We provide a proof-of-concept implementation of an SDN-enabled edge system and execute experiments measuring the performance. We show that modern edge network devices (smartphones) can act as controllers if appropriately programmed. Even for the simple scenario with four devices and three wireless links, the average delay of managing a device can significantly change for different controller placement strategies (up to 25% difference).
- *Emulation Analysis - Overheads*. We perform large-scale emulations of SDN-enabled edge networks with hundreds of nodes. Our emulations are based on a commercial controller implementation (ONOS [8]) which makes them comparable to previous studies [10]. We find that the two types of overheads (inter-controller and controller-node traffic) can be at the same order of magnitude (up to a few Mbps each). We fit the measured data to show that the inter-controller traffic closely resembles a linear function to the controller load.
- *Optimization Algorithms*. We formulate the controller placement problem for two different objectives; minimization of delay and overheads. We propose exact and approximate algorithms to optimize the two objectives using linearization and supermodular techniques.
- *Evaluation Results*. We evaluate the proposed controller placement algorithms using two real network topologies. We find that our approach performs near to optimal and better than state-of-the-art methods. Our evaluation code is publicly available [13] for the benefit of research community.

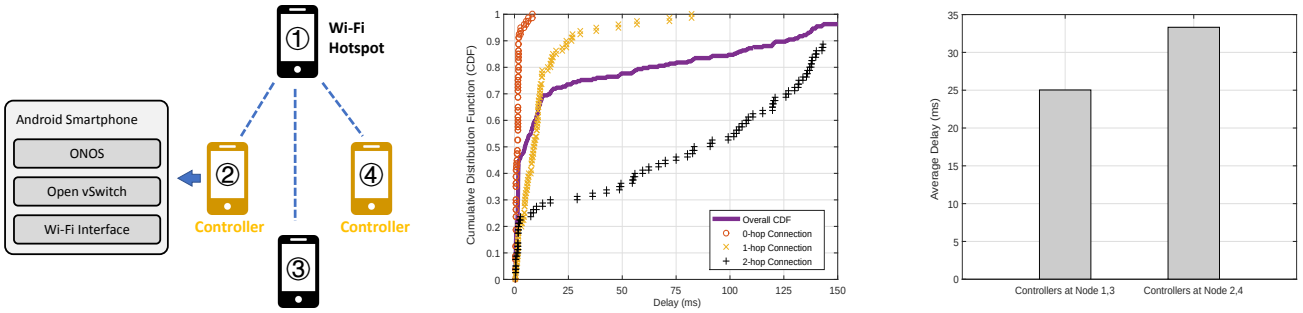
The rest of the paper is organized as follows. Section II presents our experimentation and emulation results. Guided by these findings, we model the controller placement problem in edge networks in Section III. In Section IV, we present optimal and approximate solution algorithms for small- and large-scale problem instances respectively. Section V presents the evaluation of our proposed algorithms, while Section VI reviews our contribution compared to related works. We conclude our work in Section VII.

II. Experimentation & Emulation Analysis

In this section, we present experimentation and emulation results using commercial controller and data plane implementations. The results provide insights about the delay and overheads of multi-controller edge systems which will be used in modeling the controller placement problem in the next section.

A. Experimentation Results for Management Delay

Testbed Set-up. In this subsection, we set-up a testbed of a multi-controller edge system using off-the-shelf network devices. Specifically, we deploy four Nexus 4 Android smartphones to form a wireless network as it is depicted in Figure 2(a). The first smartphone works as an access point (hotspot) to provide the remaining three smartphones with WiFi connections. This represents a common edge network setting, where a node can either establish multihop connections to backbone networks, or exchange data with its peer in a D2D fashion.



(a) Testbed topology. (b) CDF of delays. (c) Comparison of two placement schemes.

Fig. 2. (a) Testbed of a multi-controller edge system built from smartphones that enable Open vSwitch and ONOS. (b) CDF of controller-node communication delays. (c) Comparison of average delays under different controller placements.

Besides, smartphones are representative devices widely used in edge networks. Due to their constraints of calculating and storage, our testbed shows a challenging scenario that is worthy to investigate.

We take several steps to make the network SDN-enabled. In each smartphone, we create a *chroot* environment to install the Ubuntu system running with its original Android system at the same time. By this, we are able to install popular SDN-related software in the smartphone. First, we make all devices working as data plane nodes by installing Open vSwitch [11]. This creates a virtual switch that supports SDN in each smartphone. Second, we deploy SDN controllers in Node 2 and Node 4. Though constrained in resources, the smartphone is still capable enough to run a controller instance, such as ONOS. ONOS is designed particularly for scalability and permits multiple controllers working together in the form of a *controller cluster*. Then, we establish a connection between the two controllers and assign each smartphone to its nearest controller with respect to the hop count length distance metric.

Measurement Methodology. We mainly take measurements on the network delay between the controller and data plane nodes. The devices are placed within an empty lab room, and distance over each wireless link is 2 meters. One frequent and important interaction between a controller and a data plane node is the request and reply of flow statistics. Therefore, we measure the delay at controller nodes, by analyzing the interval between sending such an OpenFlow request message and receiving its corresponding reply. We keep capturing messages since the cluster reaches the steady state and collect 250 measurements. Figure 2(b) shows the cumulative distribution function of the delays we measured. The average value is tens of milliseconds which is comparable or higher than the delay reported in typical wired networks [9]. From the CDF plot, we notice that the variance is large, corresponding to the relatively unstable wireless links. It is common for the delay to go even beyond 100 milliseconds.

We also notice that the placement of controllers is important, because the delay relies highly on the distance between the data plane node and its controller. For example, Node 2 and Node 4 contain controllers locally, while Node 1 and Node 3 have one-hop and two-hop connections to the controller, respectively. As a result, drawn separately in Figure 2(b), local connection shows almost zero delays while the one-hop and two-hop connections show notable delays. To further demonstrate this, we move ONOS controllers from nodes 2 and 4 to nodes 1 and 3. If we still assign each data plane node to its nearest controller, we can find that the average delay is 25% lower, as it is depicted in Figure 2(c).

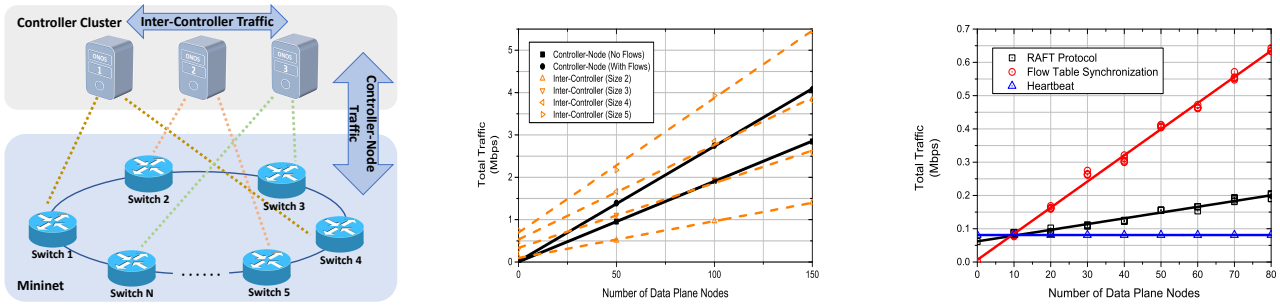
Main Takeaways. *Modern edge network devices (smartphones) can act as controllers to manage other devices. The management delay can significantly change for different controller placement strategies (up to* 25% *difference in our testbed).*

B. Emulation Results for Control Overheads

In the previous subsection, we focused on the delay required to manage the edge nodes. In this subsection, we will analyze another important factor; the overheads of SDN control.

Types of Control Overheads. When considering the SDN control overheads, people usually refer to the traffic between the controllers and data plane nodes. Namely, controllers and nodes exchange various messages through a specific protocol, which is OpenFlow in most cases, including periodic heartbeat messages and statistic requests/replies. It is intuitive that such overheads grow when the network scales up. Moreover, there are also overheads related to the routing of packet flows. When a new flow is generated and a node receives packets that cannot be matched in its forwarding table, it will report to the controller through *PacketIn* messages. On the contrary, the controller may install new forwarding rules to nodes using *FlowMod* messages. Therefore, the overhead is also influenced by the number of flows. The more frequently new flows emerge, the larger overhead is needed to install forwarding rules.

If a cluster of controllers, rather than only one controller, is deployed, a second kind of overheads will be generated, which is the state synchronization between different controllers. Among controllers, one or more consensus algorithms must be run to share the network topology and flow tables in each controller's domain, in order to reach a consistency. It can be expected that the more information they should share, and the more members they should share with, the larger such overheads are.



(a) Emulation setting. (b) Controller-node vs inter-controller overheads. (c) Impact of nodes on different messages.

Fig. 3. (a) Ring topology in emulations. (b) Controller-node overheads (solid line) and inter-controller overheads (dot lines) in a network with different numbers of nodes. (c) Breakdown of inter-controller overheads for 3 controllers and different numbers of nodes.

Emulation Set-up. Since both the types of control overheads depend heavily on the scale of the network, we need to deploy many more nodes than what we have in our testbed if we wish to analyze them. A more accessible way to take large-scale measurements is by running emulations on a virtual network generated by Mininet [14]. This method allows us to test networks with hundreds of nodes and several controllers using a common CPU machine. To be consistent with the previous subsection, we pick again ONOS as our controller instance and run it along with Mininet. Specifically, we create a virtual network with ring topology and evenly assign nodes to the placed controllers, as shown in Figure 3(a). All controllers run a simple application; reactive forwarding.

Measurement Methodology. In order to show the impact of the scale of the network, we take measurements on both types of control traffic with different number of nodes in the virtual network. Figure 3(b) verifies the intuition about both types of traffic's growth when the network scales up. What is more, we also consider other factors that have impact. For controller-node traffic, we create large amount of one-hop *iperf* [15] flows randomly, with a fixed rate (0.1 flows per second for each node). For inter-controller traffic, we deploy clusters of different sizes, i.e. number of controllers in the cluster. According to Figure 3(b), in all of these situations, the two types of overheads are at the same order of magnitude (up to a few Mbps each). This fact means that both of them are important when deciding the controller placement. We also note that the inter-controller traffic is not affected by the number of flows and this is because of the reactive forwarding application we chose.

Next, we elaborate on the inter-controller traffic overhead. In ONOS, each pair of controllers exchange periodically heartbeat messages of fixed number and sizes. Besides, there are two more consensus algorithms running [10]. The first one is anti-entropy gossip protocol [16]. With an interval, a controller sends the information (network topology, flow tables, etc.) within its domain (nodes assigned to it) to a random peer controller. Therefore, this overhead is proportional to the domain size (also called load) of this controller. The second algorithm is RAFT [17]. ONOS uses it to synchronize controller-node assignments within the cluster, generating an overhead also related to the controller load. Specifically, RAFT has a leader-follower mechanism. The controllers periodically hold elections, which can be seen as an additional constant overhead. We make separated measurements on these different components. Results in Figure 3(c) verify and quantify above analysis that *with the increasing number of nodes, the heartbeat overhead is constant. The flow table synchronization overhead (majority of anti-entropy protocol) is linear. The RAFT overhead is a combination of these two types.*

Main Takeaways. *The two types of overheads (inter-controller and controller-node traffic) are at the same order of magnitude in representative scenarios (up to few Mbps each). The traffic exchanged between a pair of controllers can be modeled by two terms; (i) a constant term (heartbeat and part of RAFT messages) and (ii) a term that increases linearly to the load of controllers (flow synchronization and part of RAFT messages).*

III. Modeling the Controller Placement Problem

In this section, we build upon the experimentation and emulation results of the previous section to define the controller placement problem in edge networks.

We consider a general (abstract) model representing an *SDN-enabled edge network* as it is depicted in Figure 4. The network contains a diverse set $\mathcal{N}$ of N *edge nodes* such as access switches, cellular base stations, fog nodes with storage and processing capabilities, wireless access points and IoT devices. Without lose of generality, the nodes generate new flows with uniform rate, normalized to one. A controller is located at the cloud and connects to the edge nodes through in-band or out-of-band control channels.

The network operator may decide to place additional controllers to some of the edge nodes. Placing a controller to an edge node requires to locally install and run a controller implementation software such as OpenDaylight [7] or ONOS [8]. Due to limited resources, not all the edge nodes may be capable of hosting a controller. To model these cases, we denote by $\mathcal{N}_h \subseteq \mathcal{N}$ the subset of nodes that can play the role of host for a controller, where $N_h = |\mathcal{N}_h|$.

The placed controllers can expedite the management of the resources of the edge nodes, since they are in closer proximity to them than the controller located at the cloud is. However, they also induce overheads due to control messaging

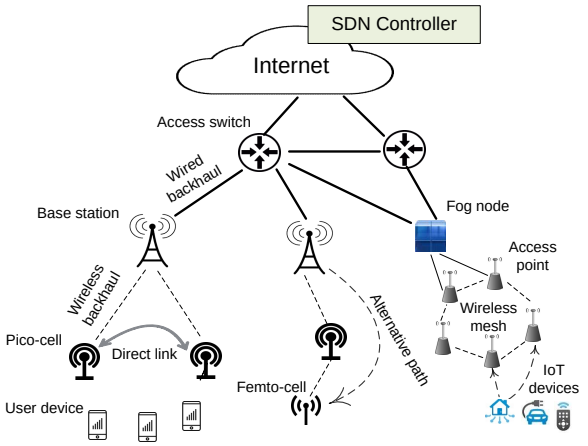


Fig. 4. An example of an SDN-enabled edge network.

between the controllers and nodes for statistic collection as well as between the controllers themselves for synchronization purposes. This gives rise to the following *tradeoff*:

On the one hand, the controllers should be close to the nodes they serve to reduce the delay of their management and the overhead of statistic collection (*assignment cost*). On the other hand, they should be close to each other to quickly synchronize their states with the lowest possible overhead (*synchronization cost*).

To study this tradeoff, we introduce the binary optimization variable $x_n \in \{0,1\}$ that indicates whether a controller is placed at node $n \in \mathcal{N}$ ($x_n = 1$) or not ($x_n = 0$). These variables constitute the *controller placement policy*:

$$\boldsymbol{x} = (x_n \in \{0,1\} \ : n \in \mathcal{N}) \ . \tag{1}$$

Clearly, we cannot place controllers at the nodes that are not powerful enough to host them:

$$x_n = 0, \ \forall n \notin \mathcal{N}_h \ . \tag{2}$$

We also introduce the binary optimization variable $y_{nm} \in \{0,1\}$ that indicates whether node $n \in \mathcal{N}$ is assigned to the controller at node $m \in \mathcal{N}$ ($y_{nm} = 1$) or not ($y_{nm} = 0$). Similarly, $y_{nc} \in \{0,1\}$ indicates the assignment of node n to the controller located at the cloud. These variables constitute the *assignment policy* of the operator:

$$\boldsymbol{y} = (y_{nm} \in \{0,1\} \ : n \in \mathcal{N}, m \in \mathcal{N} \cup \{c\}) \ . \tag{3}$$

Since every node needs to be assigned to a controller, we require that:

$$\sum_{m \in \mathcal{N} \cup \{c\}} y_{nm} = 1, \ \forall n \in \mathcal{N} \ . \tag{4}$$

In addition, we require that a controller must be placed at node m in order for node n to be able to assign to it:

$$y_{nm} \leq x_m, \ \forall n, m \in \mathcal{N} \ . \tag{5}$$

The assignment of a node to a controller induces a cost which can capture the management delay and overhead. As we showed in Figure 2(c), this cost increases with the distance in the topology between controller and node, and can be significant if the respective links are wireless. In our model, we denote by $d_{nm} \geq 0$ the cost of assigning node $n \in \mathcal{N}$ to the controller at node $m \in \mathcal{N}$. Similarly, we denote by d_{nc} the cost of assigning node $n \in \mathcal{N}$ to the controller that is located at the cloud. Typically, the cost of a node assignment to the cloud controller is large due to the long distance.

With the above notation and constraints, the total assignment cost can be expressed as follows:

$$J_a(\boldsymbol{y}) = \sum_{n \in \mathcal{N}} \sum_{m \in \mathcal{N} \cup \{c\}} y_{nm} d_{nm} \ . \tag{6}$$

The next step is to model the synchronization cost between the controllers. As we showed in Figure 3(c), each pair of controllers exchanges messages with constant rate as well as messages with rate that depends on the controller's load. The latter means that the more nodes are assigned to a controller the more messages it sends to the rest controllers. We model the respective synchronization costs as follows. For the messages exchanged at a constant rate, we denote by $w_{ml}^{con} \geq 0$ the respective cost between controllers at nodes m and l. For the messages exchanged at a rate that depends on the controller load, we denote by $w_{ml}^{dep} \geq 0$ the respective cost between controllers at nodes m and l for each node assigned to controller m. The total (across all controller pairs) synchronization cost can be expressed as follows:

$$J_s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{m \in \mathcal{N} \cup \{c\}} \sum_{l \in \mathcal{N} \cup \{c\}} x_m x_l \big(w_{ml}^{con} + w_{ml}^{dep} \sum_{n \in \mathcal{N}} y_{nm} \big). \tag{7}$$

where, with *a slight abuse of notation*, we used the terms x_m and x_l for m and l equal to c in the above summation. These terms are set to one to indicate that a controller is placed at the cloud[2].

The network operator that decides the placement ideally would like to find the right *balance* between the assignment and synchronization costs (J_a and J_s) depending on its preferences. To model this, we introduce the weight value $\gamma \geq 0$ and define the balanced function:

$$J_b(\boldsymbol{x}, \boldsymbol{y}) = J_a(\boldsymbol{y}) + \gamma J_s(\boldsymbol{x}, \boldsymbol{y}). \tag{8}$$

The edge controller placement (ECP, for short) can be expressed as follows:

$$\begin{aligned} &min_{\boldsymbol{x},\boldsymbol{y}} \ J_b(\boldsymbol{x}, \boldsymbol{y}) \\ &\quad s.t. \ \text{ constraints: } (1), \ (2), \ (3), \ (4), \ (5) \ . \end{aligned}$$

The above problem is challenging since it contains discrete variables and a non-linear objective function with cubic and quadratic terms (inside J_s). In fact, it is not hard to show that ECP is a generalization of the well-studied facility location problem [18], which is NP-Hard, by allowing facility opening (equivalently controller placement) costs to be non-constant (i.e., the synchronization cost depends on the distance between the placed controllers and to the cloud).

[2]We make the same abuse of notation throughout the paper.

IV. Optimization Algorithms

In this section, we address the ECP problem. We start by presenting an optimal algorithm that can be applied to small-scale problem instances. Following that, we present approximation algorithms that scale well with the size of the problem instance.

A. Small-scale optimal solution

In this subsection, we find an optimal solution by converting ECP to a Mixed Integer Programming (MIP) problem. That is a problem with linear objective function and constraints. This conversion is important since there exist various commercial solvers, such as CPLEX, that can be directly used to solve this type of problems.

To obtain the MIP formulation, we apply *standard linearization techniques* [12]. Specifically, we introduce the following two vectors of additional optimization variables:

$$\boldsymbol{\theta} = (\theta_{ml} \in \{0,1\} \ : m,l \in \mathcal{N} \cup \{c\}) \ . \tag{9}$$

$$\boldsymbol{\phi} = (\phi_{mln} \in \{0,1\} \ : m,l \in \mathcal{N} \cup \{c\},\ n \in \mathcal{N}) \ . \tag{10}$$

Then, we add the following linear constraints for θ:

$$\theta_{ml} \leq x_m,\ m,l \in \mathcal{N} \cup \{c\} \ , \tag{11}$$

$$\theta_{ml} \leq x_l,\ m,l \in \mathcal{N} \cup \{c\} \ , \tag{12}$$

$$\theta_{ml} \geq x_m + x_l - 1,\ m,l \in \mathcal{N} \cup \{c\} \ , \tag{13}$$

and the following linear constraints for ϕ:

$$\phi_{mln} \leq \theta_{ml},\ m,l \in \mathcal{N} \cup \{c\},\ n \in \mathcal{N} \ , \tag{14}$$

$$\phi_{mln} \leq y_{nm},\ m,l \in \mathcal{N} \cup \{c\},\ n \in \mathcal{N} \ , \tag{15}$$

$$\phi_{mln} \geq \theta_{ml} + y_{nm} - 1,\ m,l \in \mathcal{N} \cup \{c\},\ n \in \mathcal{N} \ . \tag{16}$$

The J_a function is already linear, so we only need to linearize the J_s function. This is possible with the new variables, as it can be written as:

$$\widehat{J}_s(\boldsymbol{\theta},\boldsymbol{\phi}) = \sum_{m \in \mathcal{N}\cup\{c\}} \sum_{l \in \mathcal{N}\cup\{c\}} \big(\theta_{ml} w^{con}_{ml} + \sum_{n \in \mathcal{N}} \phi_{mln} w^{dep}_{ml}\ \big). \tag{17}$$

Then, the MIP problem can be expressed as follows:

$$min_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{\theta},\boldsymbol{\phi}}\ J_a(\boldsymbol{y}) + \gamma \widehat{J}_s(\boldsymbol{\theta},\boldsymbol{\phi})$$
$$s.t.\ \text{ constraints: } (1)-(5) \text{ and } (9)-(16) \ .$$

The inequalities in (11)-(13) ensure that θ_{ml} will be zero if at least one (or equivalently the product) of x_m and x_l is zero; otherwise it will be one. Combined with the above, the inequalities in (14)-(16) ensure that ϕ_{mln} will be zero if at least one (or equivalently the product) of x_m, x_l and y_{nm} is zero; otherwise it will be one.

Various commercial solvers, such as CPLEX, can be directly used to solve a MIP problem. These solvers apply branch-and-bound techniques and can be quite fast for small-scale problem instances. However, in some cases, edge systems can be of extremely large scale, e.g., in IoT architectures, and hence the above MIP problem becomes extremely large, hindering the performance of such branch-and-bound or other computational methods. In the next subsection, we propose a solution method that overcomes this dimensionality problem, and hence extends the range of the systems to which our work can apply.

B. Large-scale approximate solution

In this subsection, we present a class of approximation algorithms that scale well with the size of the problem instance. We begin by showing that for a given controller placement $\boldsymbol{x}$, the optimal assignment policy $\boldsymbol{y}$ can be easily found. Specifically, we show the following lemma (proved in the appendix).

Lemma 1: For a given controller placement $\boldsymbol{x}$, the optimal assignment policy can be described by:

$$y_{nm} = \begin{cases} 1, \text{ if } m = \underset{m' \in \mathcal{N}\cup\{c\}\ :\ x_{m'}=1}{\operatorname{argmin}} \ [\ d_{nm'} + \gamma \sum_{l \in \mathcal{N}\cup\{c\}: x_l=1} w^{dep}_{m'l}\ ] \\ 0, \text{ otherwise} \end{cases} \tag{18}$$

for each $n \in \mathcal{N},\ m \in \mathcal{N} \cup \{c\}$.

The above lemma advises us to focus on the optimization of the controller placement policy. Following this intuition, we introduce the *element* X_n to denote the placement of a controller at node n which is equivalent of deciding $x_n = 1$. The set of all possible elements, also called the *ground set*, can be defined as follows:

$$G = (X_n : n \in \mathcal{N}_h) \ . \tag{19}$$

A subset of elements $X \subseteq G$ corresponds to a controller placement policy $\boldsymbol{x}$, such that $x_n = 1$ if and only if $X_n \in X$. Besides, let $\boldsymbol{x}_X$ be the binary representation of the set of elements X. Then, the objective function J_b can be expressed as a set function $f:\ 2^G \rightarrow \mathbb{R}$:

$$f(X) = J_b(\boldsymbol{x}_X, y(\boldsymbol{x}_X)) \tag{20}$$

where $y(\boldsymbol{x}_X)$ denotes the optimal assignment policy given the controller placement policy $\boldsymbol{x}_X$ based on equation (18).

Next, we consider a well-studied class of set functions called *supermodular* [19].

Definition 1: Let G be a finite set of elements (ground set). A set function $f:\ 2^G \rightarrow \mathbb{R}$ is called supermodular if for all subsets $A, B \subseteq G$ with $A \subseteq B$ and every element $i \in G \setminus B$ it holds that:

$$f(A \cup \{i\}) - f(A) \leq f(B \cup \{i\}) - f(B) \tag{21}$$

The above definition indicates that the marginal value for adding an element i in a set increases as the respective set expands. We will show that, under certain conditions on the cost values, our objective $f(X)$ can be expressed as a supermodular function. Specifically, we have the following lemma (proved in the appendix).

Lemma 2: The set function $f(X)$ defined in (20) is supermodular for the case of uniform costs $w^{dep}_{ml} = w^{dep}_{m'l'} = w^{dep}$, $\forall m,l,m',l' \in \mathcal{N}_h \cup \{c\}$.

Based on Lemma 2, the ECP problem can be casted as the *minimization of a supermodular function* f. This type of problems are usually addressed by considering their equivalent *submodular function maximization* version. That is, minimizing the supermodular function f (i.e., assignment and synchronization cost) is equivalent to maximizing the

TABLE I
APPROXIMATION RATIOS FOR NON-NEGATIVE SUBMODULAR FUNCTION MAXIMIZATION.

Bound β	Main technique	Reference
$\frac{1}{4}$	uniformly random	[20]
$\frac{1}{3}$	local search (deterministic version)	[20]
$\frac{2}{5}$	local search (randomized version)	[20]
0.41	simulated annealing	[21]
0.42	structural continuous greedy	[22]
$\frac{1}{3}$	greedy (deterministic version)	[23]
$\frac{1}{2}$	greedy (randomized version)	[23]

submodular function $\widehat{f}(X) = f^{ub} - f(X)$ (respective cost savings). Here, the constant f^{ub} indicates an upper bound to the highest possible value of $f(X)$.

Given that $\widehat{f}(X)$ is non-negative, there exist various approximation algorithms to maximize it (Table I). Here, an approximation bound β means that the ratio of the value of the approximate solution over the optimal solution value is always at least β, namely $\widehat{f}^{apx}/\widehat{f}^{opt} \geq \beta$. Therefore, we obtain the following theorem.

Theorem 1: There exists a solution to the ECP problem such that $\widehat{f}^{apx}/\widehat{f}^{opt} \geq \beta$, where $\beta \in \{\frac{1}{4}, \frac{1}{3}, \frac{2}{5}, 0.41, 0.42, \frac{1}{3}, \frac{1}{2}\}$.

The Algorithm in the last row of Table I has the best approximation bound. As it is summarized in Algorithm 1, this algorithm proceeds in N_h iterations which correspond to some arbitrary order $r_1, \dots r_{N_h}$ of the ground set G. At each iteration, two solutions A and B are maintained, initially set to $\emptyset$ and G respectively. At the n^{th} iteration, the algorithm either adds r_n to A or removes r_n from B. This decision is done randomly and greedily based on the marginal gain of each of the two options. After N_h iterations both solutions coincide, i.e., $A = B$. This is the output of the algorithm.

We emphasize that although the approximation bounds of Theorem 1 are shown for the case that w^{dep} values are identical, we make no assumptions on the values of d and w^{con} vectors. Moreover, in the next section we will show that in practice Algorithm 1 achieves a near-optimal solution even for heterogeneous w^{dep} values.

Algorithm 1: Randomized Greedy Algorithm

$A \leftarrow \emptyset, B \leftarrow G$
for $n = 1$ *to* N_h **do**
  $\Delta A \leftarrow f(A) - f(A \cup \{r_n\})$
  $\Delta B \leftarrow f(B) - f(B \setminus \{r_n\})$
  $\Delta A \leftarrow \max(\Delta A, 0)$, $\Delta B \leftarrow \max(\Delta B, 0)$
  with probability $\Delta A/(\Delta A + \Delta B)^*$ **do:**
  $A \leftarrow A \cup \{r_n\}$
  else (with probability$\Delta B/(\Delta A + \Delta B)$) **do:**
  $B \leftarrow B \setminus \{r_n\}$
end
return A (or equivalently B)

* If $\Delta A = \Delta B = 0$, then $\Delta A/(\Delta A + \Delta B) = 1$.

V. EVALUATION RESULTS

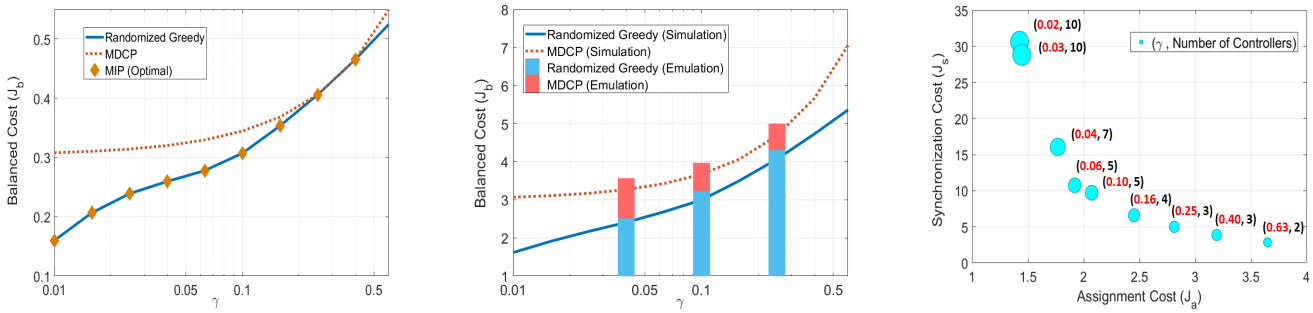
Evaluation Setup. We evaluate the proposed controller placement methods using two real network topologies. Namely, we use the MANIAC mobile ad hoc network in [24] and the Barcelona wireless mesh network in [25]. MANIAC contains only 14 nodes, which allows us to execute MIP algorithm in reasonable time. On the other hand, Barcelona contains 60 nodes, the evaluation of which enriches the results. We set the assignment and synchronization cost values of our model to be the product of the measured traffic volumes in Figure 3(b) and the network distances between the nodes. Specifically, we set $d_{nm} = 0.019 \cdot hops_{nm}$, $w^{con}_{ml} = 0.04579 \cdot hops_{ml}$ and $w^{dep}_{ml} = 0.00793 \cdot hops_{ml}$, where $hops$ indicates the number of hops of the shortest path between the respective nodes. We also add a node representing the cloud controller, connected to all other nodes with a large distance. We set this distance value as the half of the graph's diameter.

Most of the state-of-the-art methods require that the number of placed controllers is fixed and known in advance (e.g., see [9]). Therefore, it would not be fair to compare the above with our methods which optimize both the number and location of controllers. Interestingly, there is a method in the literature that explores the same solution space with our work. Namely, the MDCP method (Algorithm 3 in [26]) is a greedy procedure that places controllers based on the degrees of the nodes and the inter-controller traffic. The latter is estimated by the diameter of the topology graph, rather than based on actual measurements.

In the rest of this section, we evaluate the proposed Randomized Greedy Algorithm and compare it with MIP and MDCP methods. Our evaluation contains both simulation with MATLAB and emulation with Mininet and ONOS. The emulation procedure is the same as in Section II. All our codes are publicly available online in [13]. We expect that the reproducibility of our results will facilitate future research efforts for the benefit of research community.

Evaluation Results. Figure 5(a) is the simulation result for the MANIAC network. Using a common CPU machine, MIP takes 30 seconds to get a solution using the built-in optimizer of MATLAB, while Randomized Greedy algorithm is able to run 200 times and pick the best result as solution within 0.1 second. Overall, *Randomized Greedy has a performance very close to the optimal for all values of* γ, while the performance of MDCP drops when γ is rather small or large. In these cases, *our algorithm has up to a half lower balanced cost than MCDP*.

Figure 5(b) indicates similar conclusions in the Barcelona network. When γ is relatively large or small, the gains of our algorithm over MDCP are more pronounced. Figure 5(b) also contains emulation results. We construct the same topology in Mininet and measure the communication overheads of each pair of controllers as well as all controller-node pairs. A consistency is shown in both the value and tendency between the simulation and emulation. It implies that our model successfully captures the pattern of the controller cluster's behaviors and it is capable for providing guidance in realistic cases.



(a) Impact of γ for MANIAC network. (b) Impact of γ for Barcelona network. (c) Tradeoff between two types of cost.

Fig. 5. Balanced cost optimizations by different algorithms on the topologies of (a) MANIAC [24] and (b) Barcelona [25] networks. (c) Tradeoff between assignment and synchronization costs, as well as the number of controllers required in order to minimize the balanced cost.

We further analyze the Barcelona network simulation result in Figure 5(c), which is the tradeoff between costs of two traffic types. In fact, it is popular for controller placement researches to choose clustering methods such as k-median [9]. An issue is that these methods do not optimize the number of required controllers, but take it as an input. As shown in the figure, the algorithm we proposed can decide the proper number of controllers to place. When operators prefer to minimize the inter-controller synchronization cost (large γ), the algorithm will select fewer nodes as controllers (e.g., 2 controllers for $\gamma = 0.63$). If there is a strict demand on controller-node assignment cost (lower γ), the number of controllers will increase (e.g., 10 controllers for $\gamma = 0.02$).

VI. Related Work

Several works have proposed SDN controller placement methods over the past five years. Most of them build upon variants of the facility location or k-median problem to optimize the delay between controllers and data plane nodes. Heller et al. [9] showed that the optimal controller placement can yield five times lower delay than a random placement, while Hu et al. [27] explored the tradeoff between delay and reliability. Subsequent works [28], [29], [30] considered also the capacities of the controllers to select the placement that load balances the controllers. More advanced algorithms that dynamically adapt the controller placement based on traffic dynamics were proposed in [31] and [32].

The aforementioned works did not consider the impact of inter-controller traffic on the efficiency of the controller placement. The first works that investigated this issue were [26], [33] and [34]. However, these interesting works did not correlate the controller placement decisions with the assignment of nodes. Besides, they proposed heuristic or pareto-optimal algorithms, while in this work we present algorithms with performance guarantees following a strong experimentation and emulation analysis.

VII. Conclusion & Future Work

In this paper, we studied the SDN controller placement problem in edge network architectures. Our work combines strong experimentation results along with valid theoretical modeling and analysis. Namely, we built a testbed of a multi-controller edge system and described the sensitivity of delay as well as the magnitude and shape of traffic overheads. Guided by these findings, we presented a methodology that yields a set of optimal controller locations and assignment of nodes to controllers. Evaluation results demonstrated the benefits of our approach over state-of-the-art methods.

In the future, we plan to analyze additional mechanisms of forming a controller cluster, such as OpenDaylight [7]. Different from ONOS, the leader-follower mechanism of RAFT takes charge and leads to a different traffic pattern. We will explore whether the leader selection brings new dimension to the controller placement problem. Another fascinating topic is to relax the assumption of constant cost parameters that is commonly used in controller placement problems (e.g., see [9], [26]-[34]). Just like the overhead analysis, we will further characterize the delay, taking more factors into consideration such as the interference and congestion of the links.

References

[1] C. Mims, Forget 'The Cloud';'The Fog'Is Tech's Future, *The Wall Street Journal*, May 2014.

[2] M. Peng, S. Yan, K. Zhang, C. Wang, "Fog Computing based Radio Access Networks: Issues and Challenges", *IEEE Network, vol. 30, no. 4, pp. 46-53*, 2016.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things", *in Proc. MCC Workshop*, 2012.

[4] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Edge Computing: Vision and Challenges", *IEEE Internet of Things Journal, vol. 3, no. 5*, 2016.

[5] D. Kreutz, F. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, S. Uhlig, "Software-Defined Networking: A Comprehensive Survey", *in Proc. of the IEEE, vol. 103, no. 1, pp. 14-76*, 2015.

[6] H. Xu, Z. Yu, C. Qian, X. Li, Z. Liu, "Minimizing Flow Statistics Collection Cost of SDN Using Wildcard Requests", *in Proc. IEEE Infocom*, 2017.

[7] https://www.opendaylight.org/

[8] http://onosproject.org/

[9] B. Heller, R. Sherwood, N. McKeown, "The Controller Placement Problem", *in Proc. HotSDN*, 2012.

[10] A. Muqaddas, A. Bianco, P. Giaccone, G. Maier, "Inter-controller Traffic in ONOS Clusters for SDN Networks", *in Proc. IEEE ICC*, 2016.

[11] https://openvswitch.org

[12] F. Glover, E. Woolsey, "Converting the 0-1 Polynomial Programming Problem to a 0-1 Linear Program", *Operations Research, vol 22, no. 1, pp. 180-182*, 1974.

[13] Publicly available code: https://www.dropbox.com/s/w1vc04zmj5z5abz/infocom18_code.zip?dl=0

[14] B. Lantz, B. Heller, N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-defined Networks", *in Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, ACM, pp. 19*, 2010.

[15] https://iperf.fr/

[16] R. Van Renesse, D. Dumitriu, V. Gough, C. Thomas, "Efficient Reconciliation and Flow Control for Anti-entropy Protocols", *in Proc. ACM Workshop on Large-Scale Distributed Systems and Middleware*, 2008.

[17] D. Ongaro and J. K. Ousterhout, "In Search of an Understandable Consensus Algorithm", *in USENIX Annual Technical Conference*, 2014.

[18] D. Shmoys, E. Tardos, and K. Aardal, "Approximation Algorithms for Facility Location Problems", *in Proc. ACM STOC*, 1997.

[19] V.P. Il'ev, "An Approximation Guarantee of the Greedy Descent Algorithm for Minimizing a Supermodular Set Function", *Discrete Applied Mathematics, vol. 114, no. 1-3, pp. 131-146*, 2001.

[20] U. Feige, V. Mirrokni, J. Vondrak, "Maximizing Non-monotone Submodular Functions", *in Proc. IEEE FOCS*, 2011.

[21] S.O. Gharan, J. Vondrak, "Submodular Maximization by Simulated Annealing", *in Proc. ACM/SIAM SODA*, 2011.

[22] M. Feldman, J. Naor, R. Schwartz, "Nonmonotone Submodular Maximization via a Structural Continuous Greedy Algorithm", *in Proc. ICALP*, 2011.

[23] N. Buchbinder, M. Feldman, J. Naor, R. Schwartz, "A Tight Linear Time (1/2)-Approximation for Unconstrained Submodular Maximization", *in Proc. IEEE FOCS*, 2012.

[24] A. Hilal, J.N. Chattha, V. Srivastava, M.S. Thompson, A.B. MacKenzie, L.A. DaSilva, P. Saraswati, CRAWDAD dataset vt/maniac: http://crawdad.org/vt/maniac/20081101, https://doi.org/10.15783/C78W2V

[25] A. Neumann, E. López, L. Navarro, "An evaluation of BMX6 for Community Wireless Networks", *CNBuB*, 2012.

[26] Z. Su, M. Hamdi, "MDCP: Measurement-Aware Distributed Controller Placement for Software Defined Networks", *in Proc. IEEE ICPADS*, 2015.

[27] Y. Hu, W. Wendong, X. Gong, X. Que, C. Shiduan, "Reliability-aware Controller Placement for Software-defined Networks", *in Proc. IFIP/IEEE IM*, 2013.

[28] G. Yao, J. Bi, Y. Li, L. Guo, "On the Capacitated Controller Placement Problem in Software Defined Networks", *IEEE Communications Letters, vol. 18, no. 8, pp. 1339-1342*, 2014.

[29] A. Sallahi, M. St-Hilaire, "Optimal Model for the Controller Placement Problem in Software Defined Networks", *IEEE Communications Letters, vol. 19, no.1, pp. 30-33*, 2015.

[30] Y. Jimenez, C. Cervello-Pastor, A.J. Garcia, "On the Controller Placement for Designing a Distributed SDN Control Layer", *in Proc. IFIP Networking*, 2014.

[31] M.F. Bari, A.R. Roy, S.R. Chowdhury, Q. Zhang, M.F. Zhani, R. Ahmed, R. Boutaba, "Dynamic Controller Provisioning in Software Defined Networks", *in Proc. IEEE CNSM*, 2013.

[32] Md.T.I. ul Huque, W. Si, G. Jourjon, V. Gramoli, "Large-Scale Dynamic Controller Placement", *IEEE Transactions on Network and Service Management vol. 14, no. 1, pp. 63-76*, 2017.

[33] S. Lange, "Heuristic Approaches to the Controller Placement Problem in Large Scale SDN Networks", *IEEE Transactions on Network and Service Management, vol. 12, no. 1, pp. 4-17*, 2015.

[34] T. Zhang, A. Bianco, P. Giaccone, "The Role of Inter-controller Traffic for Placement of Distributed SDN Controllers", *in Proc. IEEE NFV-SDN*, 2016.

APPENDIX
PROOF OF LEMMA 1

Consider an optimal assignment policy $\mathbf{y}^o$ with a node n^o being assigned to the controller at a node m^o, i.e., $y_{n^o m^o} = 1$. Clearly, it should be $x_{m^o} = 1$. Let us assume that there is another node $m^h \neq m^o$ such that $x_{m^h} = 1$ and:

$$d_{nm^h} + \gamma \sum_{l \in \mathcal{N} \cup \{c\} : x_l = 1} w^{dep}_{m^h l} < d_{nm^o} + \gamma \sum_{l \in \mathcal{N} \cup \{c\} : x_l = 1} w^{dep}_{m^o l} \ . \tag{22}$$

By reassigning node n^o to m^h instead of m^o the assignment cost is reduced by $d_{nm^o} - d_{nm^h}$. At the same time, the synchronization cost is reduced by:

$$\sum_{l \in \mathcal{N} \cup \{c\} : x_l = 1} w^{dep}_{m^o l} - \sum_{l \in \mathcal{N} \cup \{c\} : x_l = 1} w^{dep}_{m^h l} \tag{23}$$

since the load-independent part of the synchronization cost is not affected by the above reassignment. Hence, the value of the objective function J_b is reduced by:

$$d_{nm^o} + \gamma \sum_{l \in \mathcal{N} \cup \{c\} : x_l = 1} w^{dep}_{m^o l} - d_{nm^h} - \gamma \sum_{l \in \mathcal{N} \cup \{c\} : x_l = 1} w^{dep}_{m^h l} \overset{(22)}{>} 0 \tag{24}$$

This contradicts our assumption of optimality of $\mathbf{y}^o$.

APPENDIX
PROOF OF LEMMA 2

Since the positively weighted sum of supermodular functions is also supermodular it suffices to show that each of the following three functions is supermodular.

$$f_{s,con}(X) = \sum_{m \in \mathcal{N} \cup \{c\}} \sum_{l \in \mathcal{N} \cup \{c\}} \mathbb{1}_{\{X_m \in X\}} \mathbb{1}_{\{X_l \in X\}} w^{con}_{ml} \tag{25}$$

$$f^n_{s,dep}(X) = \sum_{m \in \mathcal{N} \cup \{c\}} \sum_{l \in \mathcal{N} \cup \{c\}} \mathbb{1}_{\{X_m \in X\}} \mathbb{1}_{\{X_l \in X\}} w^{dep} y(\boldsymbol{x}_X)_{nm} \tag{26}$$

$$f^n_a(X) = \sum_{m \in \mathcal{N} \cup \{c\}} y(\boldsymbol{x}_X)_{nm} d_{nm} \tag{27}$$

Here, $f_{s,con}(X)$ denotes the synchronization cost that is independent of the assignment of nodes to controllers. Functions $f^n_{s,dep}(X)$ and $f^n_a(X)$ denote the rest part of synchronization cost and the assignment cost for a given node $n \in \mathcal{N}$.

Let us consider two placement sets A and B where $A \subset B \subset G$. We add an element $X_k \in G \setminus B$ to both placement sets. In other words, we place a controller at node k.

1) For the function $f_{s,con}$ and the placement set A, the marginal value of element X_k is:

$$\sum_{l \in \{l' : X_{l'} \in A\} \cup \{c\}} w^{con}_{kl} + \sum_{m \in \{m' : X_{m'} \in A\} \cup \{c\}} w^{con}_{mk} \ . \tag{28}$$

Since the above value increases if we replace A with $B \supset A$, the function $f_{s,con}$ is supermodular.

2) For the function $f^n_{s,dep}(X)$ and any placement set, the marginal value of element X_k is w^{dep} (which is independent of the assignment policy). Hence, the function $f^n_{s,dep}(X)$ is modular, which is a special class of supermodular functions.

3) For the function $f^n_a(X)$, we distinguish between two cases. **In the first case**, according to the placement set B, node n is assigned to the controller at node j where $d_{nj} \leq d_{nk}$. Then, the marginal value of element X_k is zero. For the placement set A, node n is assigned to the controller at node j'. If $d_{nj'} \leq d_{nk}$ then the marginal value is again zero. However, if $d_{nj'} > d_{nk}$ then the marginal value is $d_{nk} - d_{nj'} < 0$. **In the second case**, according to the placement B, node n is assigned to a controller at node j where $d_{nj} > d_{nk}$. Then, the marginal value of element X_k is $d_{nk} - d_{nj}$. For the placement set A, node n is assigned to a controller at node j' where it must be $d_{nj'} \geq d_{nj}$ since $A \subset B$. Hence, the marginal value is $d_{nk} - d_{nj'} \leq d_{nk} - d_{nj}$.