


Article

SDN Controller Placement and Switch Assignment for Low Power IoT

Kostas Choumas , Dimitris Giatsios, Paris Flegkas and Thanasis Korakis

Department of Electrical and Computer Engineering, Polytechnic school, University of Thessaly, Argonafton & Filellinon, 382 21 Volos, Greece; gidimitr@uth.gr (D.G.); pflegkas@uth.gr (P.F.); korakis@uth.gr (T.K.)

* Correspondence: kohoumas@uth.gr

Received: 21 January 2020; Accepted: 7 February 2020; Published: 13 February 2020



Abstract: Software defined networking (SDN) complements low power Internet of Things (IoT), since the former offers dynamicity and the latter is susceptible to environmental changes. The SDN controller placement refers to the selection of the IoT sensors running the controllers, while the switch assignment is the process of mapping each sensor to a controller. Both choices affect the volume of the control traffic, a significant metric in low power wireless IoT networks where bandwidth is scarce or energy consumption is important. In this paper, **we model an optimization problem for minimum control traffic, assess its complexity and devise a set of heuristic algorithms for expediting its solution.** We initially present a fast and simple heuristic algorithm, which is then extended to two iterative algorithms with even better performance at the cost of time complexity. Our simulations and testbed experimentation reveal close to optimal performance of all heuristic solutions with significantly less computation time than explicitly solving the optimization problem. At the end, we provide insights for further enhancements of these heuristics with a bias for minimum control delay.

Keywords: internet of things; software defined networking; testbed experimentation

1. Introduction

Preliminary results of this work have been presented at IEEE CCNC 2019 [1].

Software defined networking (SDN) decouples the control and data planes, transferring the control logic to the SDN controllers and leaving only the forwarding actions to the network equipment. The network devices can be switches, computers or sensors, which have to forward packets between each other according to a controller defined strategy. At first, SDN architecture relied on a single controller communicating with all devices, named SDN switches. However, this approach is not scalable and was soon outplaced by an advanced design exploiting more than one controller. According to this design, the load of the controller-to-switch (*Ctrl-Sw*) traffic is shared between the controllers; however, at the expense of extra traffic for the controller-to-controller (*Ctrl-Ctrl*) communication. The inter-controller traffic is necessary for the synchronization of the controllers.

IoT networks are dynamic and susceptible to environmental changes; thus, SDN complements IoT with its adjustable nature. Nevertheless, the volume of the SDN control traffic is critical for efficient IoT operation, since it affects the total energy consumption [2] and reduces the available bandwidth for the data traffic. In general, IoT networks suffer from limited energy and communication facilities, since they mostly rely on battery-powered wireless sensors. Thus, the minimization of the control traffic is very important for SDN-based IoT networks. This objective is significantly affected by the controller *placement*, which is the selection of the IoT nodes hosting SDN controllers, apart from being SDN switches themselves.

Starting from [3], a substantial amount of work has already been devoted to the research on the controller placement problem, considering a wide variety of objectives. The two questions typically

asked are: (i) How many controllers are required? (ii) Where should they be placed? Depending on the objective, the closely related problem of selecting the controller to assign a switch to (the switch assignment problem) might also be non-trivial. Although the existing literature mainly proposes controller placements that minimize the time delays for the *Ctrl-Sw* traffic, this paper considers the controller placement effect on the total control traffic (both *Ctrl-Ctrl* and *Ctrl-Sw*) and aims at its minimization. Additionally, from an architectural point of view, the primary concern of SDN-based IoT should be whether its controller placement is distributed or centralized, since it significantly impacts the network performance in terms of energy consumption, scalability and reliability [4].

The take-home message of this paper is that as the number of controllers increases and the placement becomes more distributed, the controllers get closer to the switches and the volume of the *Ctrl-Sw* traffic decreases. On the other hand, if the controllers are fewer and more concentrated, then the volume of the *Ctrl-Ctrl* traffic decreases. To exemplify those points, Figure 1 shows in the middle an IoT network with six sensors behaving as SDN switches. The left controller placement is more centralized, using only two controllers collocated with switches 1 and 2, while the right placement is more distributed, using three controllers placed at switches 1, 2 and 3. In both left and right placements, all switches have to communicate with one of the 2 or 3 controllers through 4 or 3 *Ctrl-Sw* (blue) channels, while the controllers need 1 or 3 *Ctrl-Ctrl* (red) channels respectively. The left centralized placement features less *Ctrl-Ctrl* but more *Ctrl-Sw* traffic compared to the right distributed placement, while the placement that minimizes the total control traffic depends on the *per-unit-load* of these two types of traffic, which are their minimum values in the simplest case of a *Ctrl-Sw* or *Ctrl-Ctrl* channel.

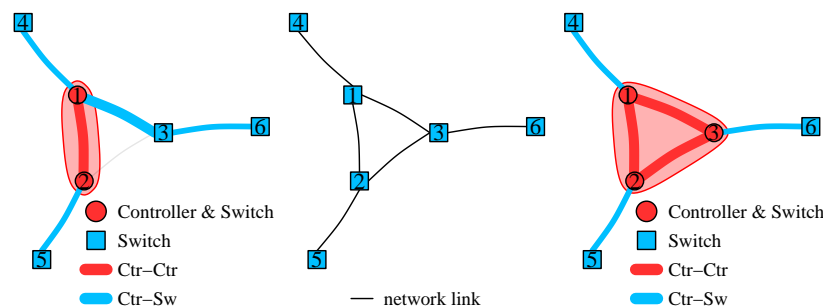


Figure 1. Toy example: The left controller placement increases the *Ctrl-Sw* traffic, while the right controller placement increases the *Ctrl-Ctrl* traffic. The line widths of the edges are proportional to the bandwidth required by the respective links.

The fourfold contributions of this paper are to:

- (i) Model the controller placement and switch assignment problem using **integer quadratic programming (IQP)** with the objective to minimize the required bandwidth for the total control traffic;
- (ii) Propose and evaluate a set of **heuristic algorithms** that expedite the aforementioned problem solution, since the IQP complexity does not scale well with the network size;
- (iii) Compare the performance of the optimal (given by IQP) and heuristic solutions, using network topologies given by the Internet Topology Zoo collection [5] (considering the graphs of this reference point as IoT network topologies);
- (iv) Provide testbed measurements for estimating the volume of both control traffic types and their per-unit-load.

In our model, the *Ctrl-Sw* (southbound) and *Ctrl-Ctrl* (east-west) protocols are OpenFlow and Raft [6,7] respectively, which are used by the state-of-the-art SDN controllers [8], such as OpenDaylight [9] and ONOS [10].

The remainder of this paper is organized as follows. Section 2 introduces related work. We present the system model and problem statement in Section 3, followed by an analysis of its optimal solution in Section 4. In Section 5, we devise a simple heuristic to expedite the problem solution, while in Section 6, we build on this and propose two iterative heuristic algorithms. Section 7 presents our experimentation results in the NITOS testbed. In Section 8, we discuss some aspects and design choices in our analysis, and provide directions for future work. The final Section 9 concludes the paper.

2. Related Work

The controller placement problem is introduced by Heller in [3], narrowing its focus to two questions: given a network topology of SDN switches, *how many controllers are needed* and *where in the topology should they be placed*? Multiple optimization goals differentiate the answers to these questions; e.g., the minimization of the average or the worst *Ctr-Sw* delay is achieved by placing the controllers according to the solution of the minimum k-median or k-center problem respectively. These optimization problems are NP-hard; thus, heuristics are exploited, such as the k-medoids algorithm [11]. The optimized criteria can also be related to the minimization of the controller load imbalance, such as the capacitated controller placement problem [12], which also considers the controller capacity and is formulated as a mixed integer linear programming (MILP) problem.

In [8], new optimized criteria are introduced by also considering the *Ctr-Ctr* traffic, except for the *Ctr-Sw* one. The goal is to minimize the reaction time perceived at the switches, having in mind that it also depends on the *Ctr-Ctr* delays, besides the *Ctr-Sw* ones. A joint study of the *Ctr-Sw* and *Ctr-Ctr* traffic overhead costs is presented in [13], minimizing the weighted sum of these two costs and two extra ones. The problem is formulated as an integer linear programming (ILP) problem and two heuristics are presented. An interesting aspect in [13] is that the dynamicity of decisions is taken into account, and switch reassignment cost is explicitly included in the model. Reassignment cost has also been considered in [14] in a virtual evolved packet core (vEPC) setting, where relocation frequency of a vEPC is among the metrics to minimize. In [11], Pareto-optimal placements are derived aiming to solve a multi-objective optimization, where one of the objectives is the minimization of the control traffic. To this end, a MATLAB framework is developed, capable of producing both an exact solution with exhaustive search and an approximate solution using Pareto simulated annealing. In [15], the learning automaton (LA)-based heuristic algorithm is introduced for controller placement. In [16], the objective of minimizing the overhead of software defined measurements is considered; a related IQP problem is formulated assuming fixed *Ctr-Ctr* costs; and an approximation algorithm with a fixed approximation ratio is devised. In [17], an approximation algorithm with a guaranteed performance bound is derived for a model similar to the one we consider in this paper; however, the proof requires that the per-unit-load of the *Ctr-Ctr* traffic does not depend on the number of flows installed in the switches, which is not the case in practice.

This is the enhanced extension of first study [1] of the controller placement and switch assignment problem, focusing exclusively on the minimization of the required bandwidth for the total control traffic. We focus exclusively on this optimization objective, since we want to analyze the effects of the contradictory tendencies of either centralizing or distributing the control plane, on the volume of the control traffic, as it is outlined in [4]. Differently to our previous study [1], we introduce two additional heuristic algorithms, with enhanced performance, at a cost of slightly more computation time (more details in Section 6). The new heuristics build upon the base heuristic presented in our previous work and apply a local search approach to discover even better placements. Moreover, we have added results for a modified version of the optimization problem with an additional constraint, which are particularly helpful as a benchmark for our heuristic estimating the number of needed controllers. Finally, a new section has been added, where we discuss some of the design choices in this paper and provide hints for further research on the topic, such as how to use our heuristic methods for minimizing both the total control traffic and the average *Ctr-Sw* delay.

3. System Model and Problem Statement

Let us assume an SDN-based IoT network, represented by a network graph $\mathcal{G} = (\mathcal{S}, \mathcal{L})$, where \mathcal{S} is the set of S switches (or sensors) and \mathcal{L} is the set of the L network links between them. Without loss of generality, we assume that the control traffic is routed through the shortest path, which is $p(s_1, s_2)$ for connecting switches $s_1, s_2 \in \mathcal{S}$, and the number of links included in this path is $|p(s_1, s_2)|$. $\mathcal{C} \subseteq \mathcal{S}$ is the subset of switches where $C = |\mathcal{C}|$ controllers are placed. From now on, we may refer to $c \in \mathcal{C}$ as a controller or the switch hosting it, interchangeably. Let $c_s \in \mathcal{C}$ denote the controller that switch $s \in \mathcal{S}$ is assigned to. Vector $\mathbf{c} = (c_s \in \mathcal{C} : s \in \mathcal{S})$ describes a controller placement and switch assignment, where each vector coordinate maps to a switch $s \in \mathcal{S}$ and the vector value indicates the corresponding controller $c_s \in \mathcal{C}$. The controller placement is given by the set of the vector values.

Our goal is to minimize the volume of the total control traffic, which is the sum of the *Ctr-Sw* and *Ctr-Ctr* traffic. The aggregated required bandwidth for the *Ctr-Sw* traffic from all network links is

$$B^S = \sum_{s \in \mathcal{S}} \sum_{l \in p(s, c_s)} b^s = \sum_{s \in \mathcal{S}} w^s b^s, \quad (1)$$

where $w^s = |p(s, c_s)|$, and b^s is the bandwidth required for the *Ctr-Sw* traffic between switch s and controller c_s . The southbound protocol dependent b^s are independent of the controller placement; thus, B^S decreases with w^s , which happens with many distributed controllers close to the switches. On the other hand, the corresponding aggregate required bandwidth for the *Ctr-Ctr* traffic is

$$B^C = \sum_{c_1 \in \mathcal{C}} \sum_{c_2 \in \mathcal{C} - \{c_1\}} \sum_{l \in p(c_1, c_2)} b^{(c_1, c_2)} = \sum_{(c_1, c_2) \in \mathcal{C}^2} w^{(c_1, c_2)} b^{(c_1, c_2)}, \quad (2)$$

where $w^{(c_1, c_2)} = |p(c_1, c_2)|$ ($w^{(c, c)} = 0$ and $b^{(c, c)} = 0$), and $b^{(c_1, c_2)}$ is the bandwidth required for the *Ctr-Ctr* traffic initiated from c_1 and sent to c_2 . The east-west protocol dependent $b^{(c_1, c_2)}$ are independent of the controller placement; thus, B^C decreases with $w^{(c_1, c_2)}$, which happens with few centralized controllers, one close to the other.

In this work, we study the optimal controller placement and switch assignment $\mathbf{c}^* = (c_s^* \in \mathcal{C}^* : s \in \mathcal{S})$ for minimum control traffic, where \mathcal{C}^* gives the optimal placement of C^* controllers. The solution of this problem is defined as

$$\mathbf{c}^* = \arg \min_{\mathbf{c}} (B^S + B^C) = \arg \min_{\mathbf{c}} \left(\sum_{s \in \mathcal{S}} w^s b^s + \sum_{(c_1, c_2) \in \mathcal{C}^2} w^{(c_1, c_2)} b^{(c_1, c_2)} \right). \quad (3)$$

Note that we can alternatively choose the weights w to reflect the per goodput byte consumed energy for a transmission along the path between the switch (or controller) and the (other) controller. In this way, we seek to minimize the total energy consumed for the control traffic. The mathematical formulation is exactly the same in both cases, so throughout this paper we consider the weights to represent shortest path lengths for simplicity.

Finally, we make the following remarks, identifying the per-unit-load of both types of traffic. More specifically, we assume (and validate later in our experimentation, presented in Section 7) that:

Remark 1. The required bandwidth for the *Ctr-Sw* traffic exchanged between a switch and its controller is proportional to the number of flows existing in this switch.

Remark 2. The required bandwidth for the *Ctr-Ctr* traffic exchanged between two controllers and initiated from one of these two is proportional to the number of switches assigned to this controller.

According to Remark 1, if β^s is the required bandwidth for a flow, then $b^s = f^s \beta^s$, $\forall s \in \mathcal{S}$, where f^s is the number of flows existing in switch s . We also assume that $f = \sum_{s \in \mathcal{S}} f^s / S$ denotes the average number of flows per switch. Moreover, in line with Remark 2, if β^c is the volume of the *Ctr-Ctr*

traffic for each assigned switch, then $b^{(c_1, c_2)} = y^{c_1} \beta^c$, $\forall (c_1, c_2) \in \mathcal{C}^2 : c_1 \neq c_2$, where $y^c = \sum_{s \in \mathcal{S} : c_s = c} 1$ denotes the number of switches assigned to controller c . As follows, the problem of Equation (3) can be rewritten as

$$\mathbf{c}^* = \arg \min_{\mathbf{c}} \left(\sum_{s \in \mathcal{S}} w^s f^s \beta^s + \sum_{(c_1, c_2) \in \mathcal{C}^2} w^{(c_1, c_2)} y^{c_1} \beta^{c_2} \right). \quad (4)$$

4. Problem Solution

4.1. Insights from the Closed form Solution for Mesh Networks

Let us consider a mesh network where all switches have the same number f of flows and there is a link between each pair of switches. We search for the optimal controller placement and switch assignment; that is, the solution to the problem of Equation (4). Because of the mesh network symmetry, only the number of controllers effects the solution efficiency and not their placement. Thus, after finding the optimal size C^* of the controller set, their placement can be done randomly. In addition, the switches can be randomly assigned to the controllers, keeping in mind that each controller must control at least the switch that it is collocated with.

From Equation (1) and Remark 1, we have

$$B_{\text{mesh}}^S = \sum_{s \in \mathcal{S} - \mathcal{C}} f^s \beta^s = \sum_{s \in \mathcal{S} - \mathcal{C}} f \beta^s = (S - C) f \beta^s. \quad (5)$$

Moreover, each controller c is one-hop away from the other controllers, and it is responsible for $\sum_{s: c_s = c} 1$ switches (\sum_s , \sum_c and $\sum_{(c_1, c_2)}$ are equivalent to $\sum_{s \in \mathcal{S}}$, $\sum_{c \in \mathcal{C}}$ and $\sum_{(c_1, c_2) \in \mathcal{C}^2}$ respectively). Thus, $\sum_c \sum_{s: c_s = c} 1 = S$, since all controllers are responsible for all switches. As follows, from Equation (2) and Remark 2, we have

$$B_{\text{mesh}}^C = \sum_{(c_1, c_2): c_1 \neq c_2} \sum_{s: c_s = c_1} \beta^c = \sum_{c_1} \sum_{s: c_s = c_1} \sum_{c_1 \neq c_2} \beta^c = \sum_{c_1} \sum_{s: c_s = c_1} (C - 1) \beta^c = S(C - 1) \beta^c. \quad (6)$$

The number C^* minimizing the sum $B_{\text{mesh}}^S + B_{\text{mesh}}^C = (S - C) f \beta^s + S(C - 1) \beta^c = S(f \beta^s - \beta^c) + C(S \beta^c - f \beta^s)$ is equal to

$$C^* = \begin{cases} 1, & \text{if } S \beta^c - f \beta^s \geq 0 \Rightarrow f \beta^s / \beta^c \leq S \\ S, & \text{if } S \beta^c - f \beta^s < 0 \Rightarrow f \beta^s / \beta^c > S \end{cases}. \quad (7)$$

This is a toy example that clearly presents an outcome of this study; namely, the relation between C^* , the fraction $f \beta^s / \beta^c$ and the network size S . The optimal number of controllers C^* increases with the *Ctrl-Sw* traffic ($f \beta^s$) and decreases with the *Ctrl-Ctrl* traffic (β^c) and the network size S . Next, we formulate the same problem for various topologies and examine optimal and heuristic solutions.

4.2. Integer Quadratic Programming (IQP) for Optimal Solution

Let us examine the case of a general network topology \mathcal{G} , where, similar to our toy example, all switches have the same number of flows f . We make the simplifying assumption that all switches feature the same number of flows, as a first step to approach an otherwise fairly complicated problem. This is the major compromise we make in this work, in our attempt to comprehend the nature of the problem. We also assume that both the number of flows per switch f and the network topology remain constant for the interval where the resulting placement and assignment will be applied.

The optimization problem of Equation (4) is equivalent to the following IQP problem

$$\min_{\mathbf{x}, \mathbf{y}, \mathbf{z}} f\beta^s \sum_{i=1}^S \sum_{j=1}^S w_{ij}x_{ij} + \beta^c \sum_{i=1}^S \sum_{j=1}^S w_{ij}y_i z_j \quad (8)$$

$$\begin{aligned} \text{s.t.} \quad & \sum_{j=1}^S x_{ij} = 1, & \forall i = 1, \dots, S, \\ & \sum_{i=1}^S x_{ij} = y_j, & \forall j = 1, \dots, S, \\ & z_i \geq y_i/S, & \forall i = 1, \dots, S, \\ & x_{ij}, z_i \in \{0, 1\}, y_i \in \mathbb{N} & \forall i, j = 1, \dots, S, \end{aligned} \quad (9)$$

where i and j take integer values from 1 to S , and each value corresponds to a switch $s \in \mathcal{S}$. If $x_{ij} = 1$, then switch s_i is assigned to controller c_j , which is collocated with switch s_j . Non-negative integer y_i is the number y^{c_i} of switches assigned to controller c_i . Binary $z_i = 1$ if and only if a controller is placed at switch s_i . Finally, w_{ij} is the length of the path connecting switch s_i (or controller c_i) and controller c_j .

In Equation (8), the minimized sum consists of two terms; the first corresponds to the *Ctrl-Sw* traffic (B^S) and the second one to the *Ctrl-Ctrl* traffic (B^C). The first term is the sum of the lengths of the paths connecting all switches to their controllers, multiplied by $f\beta^s$. The second term is the sum over all controller pairs of the products between the length of the path connecting them and the number of switches assigned to one of them, scaled by β^c . The first constraint restricts each switch to be assigned to only one controller, while the second and third constraint guarantee that y_i and z_i have the aforementioned meaning. Especially for the third constraint, binary variable z_i has to be minimized; thus, $z_i = 0$, if $y_i = 0$; otherwise, $z_i = 1$, since $0 \leq y_i \leq S$.

The optimal controller placement given by the solution of Problem (8)–(9) is $\mathcal{C}^* = \{s_i \in \mathcal{S} : z_i = 1\}$ and the switch assignment is $\mathbf{c}^* = (c_{s_i}^* = c_j : x_{ij} = 1)$. Given the symmetric matrix $\mathbf{w} = (w_{ij} : i, j = 1, \dots, S)$ induced by \mathcal{G} , the objective function of Equation (8) is equivalent to $\mathbf{v}^T \mathbf{Q} \mathbf{v} / 2 + \mathbf{q}^T \mathbf{v}$, where

$$\mathbf{Q} = \beta^c \begin{bmatrix} \mathbf{0}^{S^2 \times S^2} & \mathbf{0}^{S^2 \times S} & \mathbf{0}^{S^2 \times S} \\ \mathbf{0}^{S \times S^2} & \mathbf{0}^{S \times S} & \mathbf{w}^{S \times S} \\ \mathbf{0}^{S \times S^2} & \mathbf{w}^{S \times S} & \mathbf{0}^{S \times S} \end{bmatrix}, \quad \mathbf{q} = f\beta^s \begin{pmatrix} \tilde{\mathbf{w}}^{1 \times S^2} & \mathbf{0}^{1 \times S} & \mathbf{0}^{1 \times S} \end{pmatrix}, \quad \mathbf{v} = \begin{pmatrix} \tilde{\mathbf{x}}^{1 \times S^2} & \mathbf{y}^{1 \times S} & \mathbf{z}^{1 \times S} \end{pmatrix}, \quad (10)$$

$\mathbf{x} = (x_{ij} : i, j = 1, \dots, S)$, $\mathbf{y} = (y_i : i = 1, \dots, S)$ and $\mathbf{z} = (z_i : i = 1, \dots, S)$. The vectors $\tilde{\mathbf{w}}$ and $\tilde{\mathbf{x}}$ are composed of all the rows of the matrixes \mathbf{w} and \mathbf{x} respectively. The superscript indices next to the matrix symbol give the matrix dimensions, while the $\mathbf{0}$ matrix is full of zeros.

It is not hard to show that this problem is a generalization of the well-studied facility location problem, which is NP-Hard, as already observed in [17]. For large network instances, using IQP to solve the problem might take a prohibitively long time, especially considering that in a dynamic environment the solution is of value only for as long as the network topology \mathcal{G} and the average number of flows f stay constant. In the following two Sections 5 and 6, we present some heuristics which trade off optimality with more reasonable computation times.

5. Heuristic Solution

5.1. Number of Controllers by Linear Regression

As a first step for proposing heuristics for the control traffic minimization problem, we seek to obtain an expression which will yield estimates of the optimal number of controllers, given the average number of flows f and the network size S . We will denote these estimates by C^h , with the superscript h indicating that their purpose is for use in heuristic algorithms. Arguably, there are more properties of the network's topology graph which generally affect the optimal number of controllers, such as the

degree distribution. We tried, however, to obtain a relatively simple expression, even at the cost of less accurate estimates.

To do so, we solve Problems (8)–(9) for multiple network topologies, provided by the Internet Topology Zoo collection [5], which is a reference point widely used in the controller placement research. Our purpose is to extract an appropriate expression from the optimal solutions, to be used as a predictive model for the general case. We do not consider the bandwidth or latency features of these networks, but only the graphs of this collection, since our focus is on IoT networks. In our measurements, we assume that each of these networks represents the network topology \mathcal{G} .

The optimal solutions for the examined topologies clearly show a linear relationship between the fractions C^*/S and $f\beta^S/\beta^C$. Figure 2a depicts this linear relationship for 7 networks with various network sizes, and the results of linear regression for those networks. The full set of our solutions includes 135 networks with size $S \leq 30$ and confirms this relationship. The analysis of the results shows that:

- For low $f\beta^S/\beta^C$, B^C is more weighted than B^S , controllers are placed more centrally and C^*/S decreases.
- For high $f\beta^S/\beta^C$, the weight of B^S is amplified, more distributed controllers are placed and C^*/S increases.

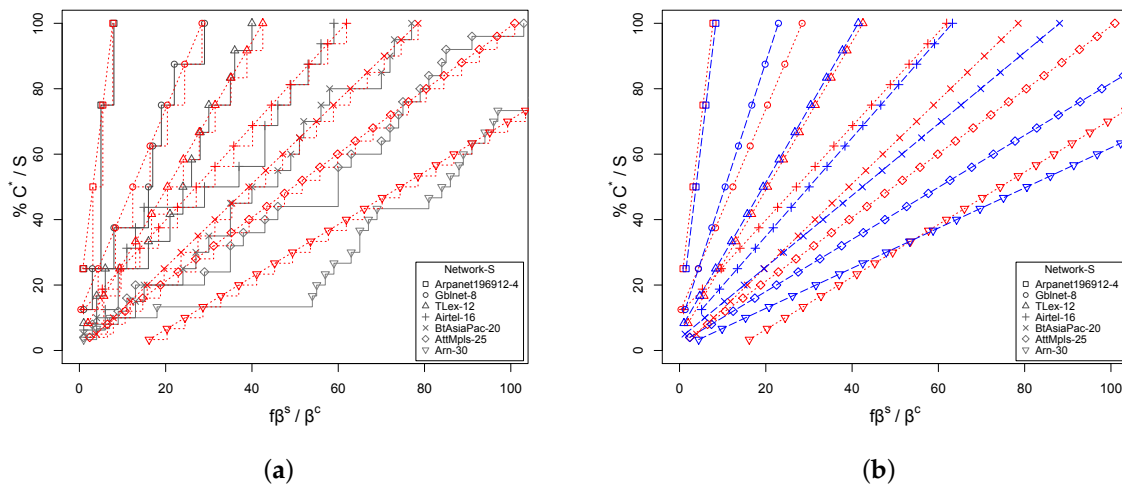


Figure 2. Linear regression between C^*/S and $f\beta^S/\beta^C$ for seven example topologies. Topology names, as listed in the Internet Topology Zoo collection, are suffixed by their size S . (a) Solid gray lines depict the relationship between C^*/S and $f\beta^S/\beta^C$. Dashed red lines depict the linear regression of this relationship (the stairstep lines depict the percentage corresponding to the integer number of controllers). (b) Dashed red lines depict the linear regression of the relationship between C^*/S and $f\beta^S/\beta^C$, as before. Blue dashed lines depict the linear models extracted by Equation (11).

The linear prediction model of the relationship between C^*/S and $f\beta^S/\beta^C$ is noted as $(C^*/S) \simeq a(f\beta^S/\beta^C) + b$. We use the optimal solutions to extract slope a and y-intercept b . The slope a decreases as the network size S increases. Their exponential relationship is illustrated by the dashed line in Figure 3a. The corresponding dashed line in Figure 3b depicts the linear relationship between y-intercept b and network size S . Using these models,

$$C^h = \left\lfloor (a(f\beta^S/\beta^C) + b)S \right\rfloor^{[1,S]}, \text{ where } a = 0.79/S^{1.43}, b = -0.003S + 0.0961 \quad (11)$$

and $\lfloor x \rfloor^{[1,S]}$ is the closest integer that is lower than x , except for the cases that x is lower than 1 or higher than S , where $x = 1$ or $x = S$ respectively. Figure 2b depicts the closeness of the linear regressions of the relationship between C^*/S and $f\beta^S/\beta^C$ to the linear models extracted by Equation (11).

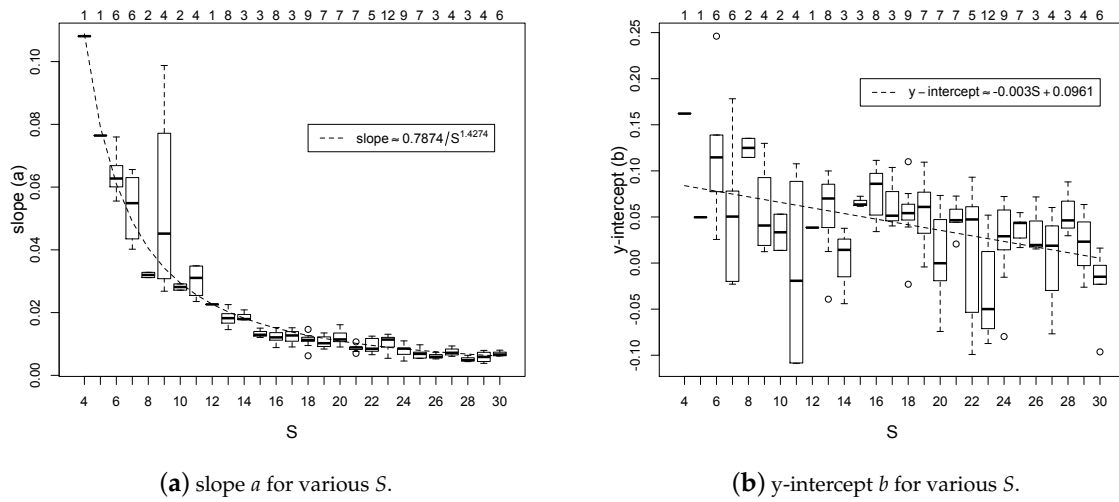


Figure 3. Slope a and y-intercept b for various networks of size $S \leq 30$, given by the Internet Topology Zoo collection. Top axis gives the number of samples for each S . The dashed lines model the regression between the boxplot medians and S .

5.2. Controller Placement Using Centrality Metric

Given this estimate of the number of controllers, C^h , the next challenge is to place them. According to our heuristic method, the C^h controllers are placed at the most “central” switches, which form the C^h set. The centrality of each switch is evaluated with the use of the betweenness [18] metric, which is based on shortest paths and gives higher value to nodes with more control over the network. In this way, the placement needs reasonable computational time, almost negligible, which scales as $O(SL + S^2 \log(S))$ using the Brandes algorithm [19].

5.3. Switch Assignment

The final challenge is to make the switch assignment. Given any controller placement \mathcal{C} in the form of a binary vector $\mathbf{z} = \{z_i = 1 : c_i \in \mathcal{C}\}$, which has 1 at the indices of the switches where controllers are placed, we get the following switch assignment sub-problem.

$$\min_{\mathbf{x}} \sum_{i=1}^S \sum_{j=1}^S (f\beta^s w_{ij} + \beta^c \sum_{m=1}^S w_{jm} z_m) x_{ij} \quad (12)$$

$$\begin{aligned} \text{s.t.} \quad & \sum_{j=1}^S x_{ij} = 1, \quad \forall i = 1, \dots, S, \\ & \sum_{i=1}^S \sum_{j=1}^S z_j x_{ij} = S, \\ & x_{ij} \in \{0, 1\} \quad \forall i, j = 1, \dots, S, \end{aligned} \quad (13)$$

where binary z_m is not a variable anymore but a given value. It is straightforward to see that this sub-problem is solvable in polynomial time. Indeed, x_{ij} are typical assignment variables, and the parenthesis in the objective function gives the assignment cost if switch i is assigned to controller j . For each switch, a simple lookup over the respective \mathcal{C} assignment costs suffices, so the overall computation scales as $O(SC)$.

In our heuristic solution, the switch assignment is given by the solution to the above sub-problem, when vector \mathbf{z} corresponds to placement \mathcal{C}^h .

5.4. Evaluation of the Heuristic Solution

Figure 4a shows the percentage increase on the total control traffic, when C^h is used instead of C^* , for the same 7 networks with Figure 2. The increase is less than 5% for the great majority of $f\beta^s/\beta^c$, with only the exception of the largest network topology “Arn” that features the highest increase close to 23%, only, however, for few values of $f\beta^s/\beta^c$. In Figure 4b, the black boxplots (labeled as “heuristic”) show the average percentage increase of the control traffic over all values of $f\beta^s/\beta^c$, for various networks. Each boxplot corresponds to a set of networks, which are grouped based on their size. For each network, we estimate the average percentage increase over all integer values of $f\beta^s/\beta^c$ between 1 and the value that both our heuristic method and the optimal solution place controllers at all network switches. The total average increase over all cases is approximately 4.5% and the highest increase is approximately 25%. The median increase of each boxplot is very low for small networks and extends up to approximately 5% for large networks.

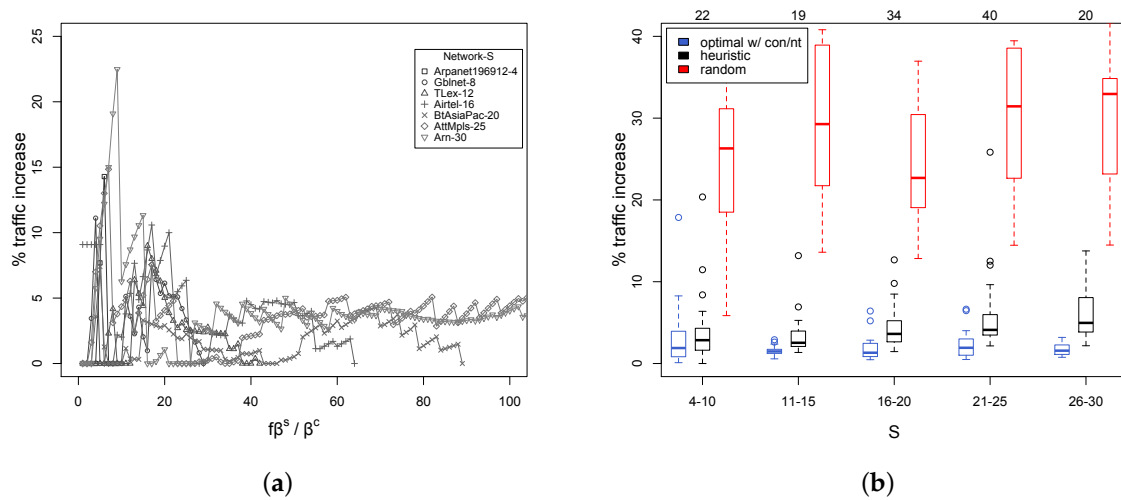


Figure 4. Percentage of control traffic ($B^S + B^C$) increase when C^h is used instead of C^* , for various networks in the Internet Topology Zoo collection. (a) Percentage of control traffic increase for various $f\beta^s/\beta^c$ in 7 example topologies. Topology names are suffixed by their size S . (b) Percentage of control traffic increase for several solutions, with respect to the optimum, averaged over all $f\beta^s/\beta^c$ values, for various networks of size $S \leq 30$. Top axis gives the number of network topology samples for each group of S .

The blue boxplots (labeled as “optimal w/ con/nt”), on the left side of the black ones, show the corresponding increase when we solve the Problem (8)–(9), however, introducing one more constraint. This constraint is

$$\sum_{i=1}^S z_i = C^h, \quad \forall i = 1, \dots, S, \quad (14)$$

which restricts the time complexity of this problem by reducing the feasible region. The solution of this new problem uses the number of controllers given by Equation (11); however, their placement and the switch assignment is estimated as in the optimal solution. We provide these results to evaluate, individually, the method for estimating the number of controllers. The method seems to be very efficient, since the total average increase over all cases is less than 2% and the highest one is approximately 18%.

Finally, the red boxplots (labeled as “random”) show the average percentage increase of the control traffic for a random placement of C^h switches, depicting how much worse the traffic increase would be if the controller placement was random, without use of centrality metrics, just by using

the controller cardinality estimate. The average increase over all network sizes for random controller selection is equal to 27% and the highest is 42%, records which are much higher than the previous ones for the C^h placement.

Figure 5 shows the relative time needed for each solution. The time needed for the proposed heuristic solution is significantly less than this of the optimal one, since solving Problem (12)–(13) is much faster than Problem (8)–(9), and the estimation of the betweenness metric for all switches needs negligible time. The black boxplots present a percentage comparison between the time requirements of both solutions. In average, the heuristic method requires 0.7% of the time needed for the optimal solution, which increases up to only 6.5% in the worst case. The blue boxplots, on top of the black ones, present the time required for solving Problem (8)–(9) with the extra constraint of Equation (14), which is between the times required for the optimal and heuristic solutions. In average, it requires 46% of the time needed for the optimal solution. Thus, even if solution of Problem (8)–(9)–(14) has slightly better performance in terms of control traffic bandwidth requirements, it needs significantly higher execution time than the proposed heuristic solution.

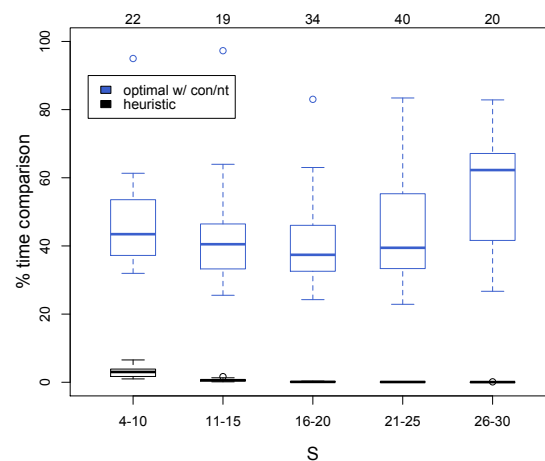


Figure 5. Time required by several solutions, as percentage fractions of the optimal solution time. Results are grouped by S . Top axis gives the number of network topology samples for each S .

Note that these solutions were derived using R [20] and CPLEX [21] in a virtual machine with 12 GB RAM and 4 of the 6 Intel Xeon CPU E5-2620 @ 2.00 GHz cores provided by the hosting HP ProLiant DL380p Gen8 physical server. Under this environment, the calculation of the heuristic and optimal solution require on average 0.002 and 78 s respectively.

6. Iterative Heuristics

Although the performance of the previous heuristic is in general satisfactory, examining a single value is obviously an extreme scenario for a more generic family of iterative heuristic solutions. We propose here two deterministic iterative algorithms examining multiple candidate placements based on a suitably defined local search procedure.

6.1. Local Search Algorithm with a Fixed Number of Controllers

A local search algorithm constitutes moving along a trajectory of feasible solutions, where each solution is a *neighbor* to the immediate previous, yielding iteratively lower costs until we reach a point where no neighboring solution exists that further lowers the cost. To move to a new solution, all neighbor solutions of the current one are examined, and the one yielding the largest decrease is selected. The obvious drawback of this algorithm is that it generally saturates at some local optimum, which might not be a global one.

A key design decision in a local search algorithm is to define what constitutes a neighboring solution. There is an underlying trade-off. If the neighborhood size is too large, a lot of time is consumed by evaluating all the neighbor solutions before each move. In the limit as the neighborhood is expanded, this approaches exhaustive search. On the other hand, if the neighborhood is too small, it is easier to get trapped in a local optimum.

In our context, we chose a simple notion of neighborhood. In particular, we consider placement \mathcal{C}^A to be a neighbor of placement \mathcal{C}^B , if and only if \mathcal{C}^A is derived from \mathcal{C}^B by moving exactly one controller to an adjacent switch not currently hosting another controller. Through this definition, we manage to always keep the number of controllers constant. We call the algorithm performing a local search starting from any initial placement *local-search-fixed*, because we keep the number of controllers invariant throughout it. Its detailed operation is presented in Algorithm 1.

One first generalization of the simple heuristic of the previous section is thus to start with the placement \mathcal{C}^h and plug it in the *local-search-fixed* algorithm, which will then perform a local search until saturation; that is, until there is no move to a neighboring placement that lowers the cost.

Algorithm 1 Local-search algorithm with fixed number of controllers (local-search-fixed).

```

1: input:  $\mathcal{S}, \mathbf{w}, \mathcal{C}^o, f$ 
2:
3:  $\mathcal{C} \leftarrow \mathcal{C}^o$ 
4:
5:  $cost \leftarrow \text{Assign-Prob}(\mathcal{C}, \mathbf{w}, f)$  (solution of Prob. (12)–(13))
6:
7: repeat
8:
9:    $\mathcal{C}' \leftarrow \mathcal{C}$ 
10:
11:   for  $c \in \mathcal{C}'$  do
12:
13:      $\mathcal{N}^c \leftarrow \{s \in \mathcal{S} | w^{(c,s)} = 1, s \notin \mathcal{C}'\}$ 
14:
15:     for  $n \in \mathcal{N}^c$  do
16:
17:        $\mathcal{C}'' \leftarrow \mathcal{C}' - \{c\} \cup \{n\}$ 
18:
19:        $cost'' \leftarrow \text{Assign-Prob}(\mathcal{C}'', \mathbf{w}, f)$ 
20:
21:       if  $cost'' < cost$  then
22:
23:          $cost \leftarrow cost''$ 
24:
25:          $\mathcal{C} \leftarrow \mathcal{C}''$ 
26:
27:       end if
28:
29:     end for
30:
31:   end for
32:
33: until  $\mathcal{C} = \mathcal{C}'$ 
34:
35: return  $cost, \mathcal{C}$ 

```

6.2. Local Search Algorithm with Variable Number of Controllers

A second generalization comes from the observation that the number of controllers derived from the regression is an estimate which does not coincide with the optimal number in many cases, but it does not generally differ from it by more than a few controllers. A natural idea is then to apply local search not only starting from the \mathcal{C}^h nodes, but to also probe lower and higher numbers of controllers. For each number, we use the betweenness centrality rank of nodes to select the initial placement, and subsequently apply the *local-search-fixed* algorithm. First, we keep reducing the number of controllers, one at a time, until we observe an increase in the returned cost. Similarly, we then increase the number of controllers from \mathcal{C}^h and upwards until an increase of cost is observed. When lowered cost values have been observed for both lower and higher than \mathcal{C}^h numbers of controllers, we just select the placement that yields the lowest cost. We call this algorithm *local-search-variable*. The related pseudocode is presented in Algorithm 2.

Note that we still examine higher numbers, even if we have found the cost to decrease with numbers lower than \mathcal{C}^h . Indeed, there is no guarantee of monotonicity of the output of the algorithm,

something we have also observed on several occasions in practice. This means that, for example, the returned cost for 10 controllers might be higher than the returned cost for both 9 and 11 controllers. A consequence of this is that we cannot be certain that we could not have encountered even lower cost values if we ignored the cost increases at the points we stopped, and continued increasing or decreasing controller cardinality. A natural question then arises, whether applying *local-search-fixed* for all possible numbers of controllers is worth the extra computation time. In our simulations, we found that this approach very rarely produced any improvement at all over *local-search-variable*, while often consuming significantly more time. The regression based estimation of C^h pays off, in saving us from valuable computation time.

Algorithm 2 Local-search algorithm with variable number of controllers (local-search-variable).

```

1: input:  $S, \mathbf{w}, C^h, f$ , betweenness-based order of  $S$ 
2:
3:
4:  $cost^0, C^0 \leftarrow \text{local-search-fixed}(S, \mathbf{w}, C^h, f)$ 
5:
6:  $C^i \leftarrow C^h$ 
7:
8:  $C' \leftarrow C^0$ 
9:
10:  $cost' \leftarrow cost^0$ 
11:
12: repeat
13:
14:    $C \leftarrow C'$ 
15:
16:    $cost \leftarrow cost'$ 
17:
18:    $C^i \leftarrow C^i - \{c \in C^i \text{ with lowest bet/ness metric}\}$ 
19:
20:    $cost', C' \leftarrow \text{local-search-fixed}(S, \mathbf{w}, C^i, f)$ 
21:
22: until  $cost' \geq cost$ 
23:
24:  $cost'' \leftarrow cost$ 
25:
26:  $C'' \leftarrow C$ 
27:
28:  $C^i \leftarrow C^h$ 
29:
30:  $C' \leftarrow C^0$ 
31:
32:  $cost' \leftarrow cost^0$ 
33:
34: repeat
35:
36:    $C \leftarrow C'$ 
37:
38:    $cost \leftarrow cost'$ 
39:
40:    $C^i \leftarrow C^i \cup \{s \in S - C^i \text{ with highest bet/ness metric}\}$ 
41:
42:    $cost', C' \leftarrow \text{local-search-fixed}(S, \mathbf{w}, C^i, f)$ 
43:
44: until  $cost' \geq cost$ 
45:
46: if  $cost > cost''$  then
47:
48:    $cost \leftarrow cost''$ 
49:
50:    $C \leftarrow C''$ 
51:
52: end if
53:
54: return  $cost, C$ 

```

6.3. Evaluation of Iterative Heuristics

In Figure 6a we can see how the two proposed iterative heuristics compare with our initial heuristic in terms of cost increase from the optimal cost value. We have clustered our results in five groups according to the number of switches of the respective topologies. While network size alone is not sufficient for characterizing the difficulty of the minimum traffic problem in hand, it provides a coarse measure for estimating it. Similar to the previous section, the percentage increase is the average over all integer values of $f\beta_s/\beta_c$ between 1 and the value that both our heuristic method and the optimal solution place controllers at all network switches.

While the median value of cost increase does not surpass 5% even for our simple initial heuristic, the importance of applying an iterative heuristic algorithm becomes greater when we look at worst case cost increases. Indeed, while in the worst case the cost increase with the initial heuristic can surpass 10%, the respective worst case increase for local-search with a fixed number of controllers is no more than 6%, and for a variable number of controllers just a little over 2%.

The results of the time required to obtain these values are depicted in Figure 6b. For topologies with a small number of switches, we observe that most of the time the optimal solution can be found even faster than the iterative heuristics, as the problem at hand is relatively easy. As the size and complexity of the graph grows, however, greater and greater time savings can be obtained by the heuristic solutions.

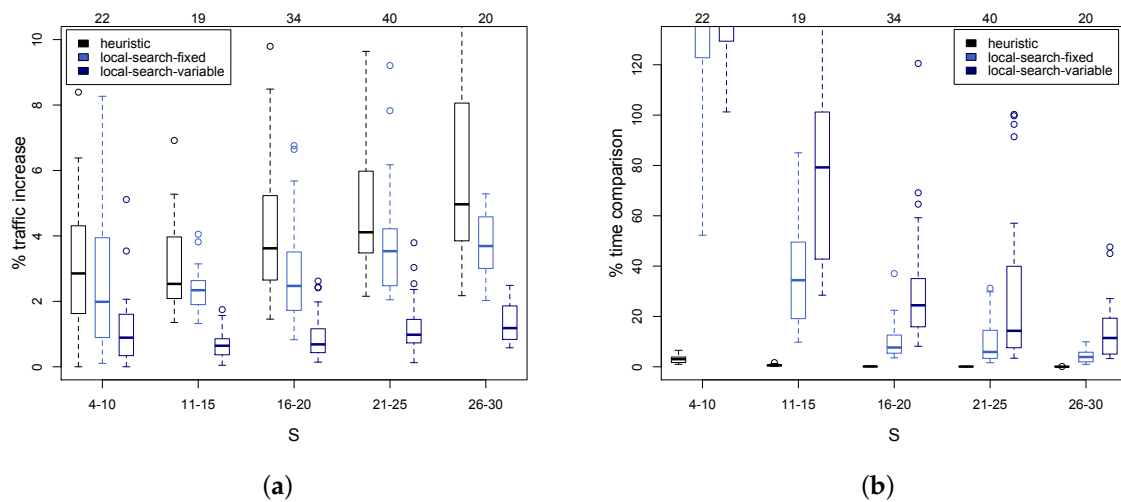


Figure 6. Performance comparison of the heuristic solutions in terms of control traffic and computation time. Results are grouped by S . Top axis gives the number of network topology samples for each S . (a) Percentage of control traffic increase for the heuristic solutions, with respect to the optimal, averaged over all $f\beta^s/\beta^c$ values. (b) Time required by the heuristic solutions, as a percentage fraction of the optimal solution time.

7. Testbed Experimentation

Our heuristics have been evaluated in the SDN-based IoT testbed of NITOS [22], which includes 30 raspberry pi 3 b+. All raspberry nodes are equipped with Open vSwitch (OvS), enabling their exploitation as OpenFlow-based SDN switches. SDN controllers are built with the assistance of the Kandoo framework [23]. Kandoo controllers are organised hierarchically, since one is chosen as the root controller and the others become the local controllers. However, the control traffic produced by both root and local controllers follow the same pattern, as we will show in Section 7.1. The hierarchy is mainly for introducing each controller to the other, since all controllers first communicate with the root controller and then they learn about each other.

7.1. Experimental Confirmation of Remarks 1 and 2

The validity of Remarks 1 and 2 is confirmed by the experimentation results presented in Tables 1 and 2. Table 1 shows the volume of the $Ctr-Sw$ traffic (measured in kbps) between a Kandoo controller and a variant number of switches, as it is estimated by *iftop*. The first and second column give the number of switches connected to the controller and the number f of flows existing at each switch. The next three columns show how much $Ctr-Sw$ traffic is produced in each direction; switch to controller and the opposite; and their sum. Finally, the last column shows how traffic increases with the number of flows. Obviously, the traffic of each switch is independent of the other switches and linearly dependent on the number of its flows. Thus, Remark 1 holds and $\beta^s \approx 1.38$ kbps.

Table 1. The *Ctr-Sw* bandwidth requirements in kbps for a Kandoo controller. Columns *sw→ctr* and *ctr→sw* refer to the traffic sent from the sw(itch) to the c(on)tr(oller) and the opposite.

# sw	# flows	<i>sw→ctr</i>	<i>ctr→sw</i>	<i>sw↔ctr</i>	Increase
1	0	1.06	2.50	3.56	-
	2	3.81	2.50	6.31	2×1.38
	3	5.19	2.50	7.69	1.38
	10	14.80	2.50	17.30	7×1.37
	20	29.40	2.50	31.10	10×1.38
10	0	10.60	25.0	35.60	-
	10	148.0	25.0	173.6	$10 \times 10 \times 1.38$
	20	294.0	25.0	311.0	$10 \times 10 \times 1.37$

Table 2. The *Ctr-Ctr* bandwidth requirements in kbps for two or three Kandoo controllers. In the left part of the table, there are two controllers, the root (c_r) and the local (c_l). In the right part, there are three controllers, which are named as root (c_r), local-1 (c_{l1}) and local-2 (c_{l2}).

# sw at	Band/th		Increase	# sw at	Bandwidth		Increase											
c_r, c_l	$c_l \uparrow$	$c_r \uparrow$	$c_l \uparrow$	$c_r \uparrow$	c_r, c_{l1}, c_{l2}	$c_{l1} \uparrow$	$c_r \uparrow$	$c_{l2} \uparrow$	$c_r \uparrow$	$c_{l2} \uparrow$	$c_{l1} \uparrow$	$c_{l2} \uparrow$	$c_r \uparrow$	$c_{l1} \uparrow$	$c_{l2} \uparrow$	$c_r \uparrow$	$c_{l2} \uparrow$	$c_{l1} \uparrow$
0,0	20	20	-	-	0,0,0	20	20	20	20	0	0	-	-	-	-	-	-	-
1,1	103	103	83	83	1,1,1	105	105	105	105	105	105	85	85	85	85	105	105	105
2,2	151	151	48	48	2,2,2	151	151	151	151	151	151	46	46	46	46	46	46	46
3,3	197	197	46	46	10,10,10	490	490	490	490	490	490	8×42	8×42	8×42	8×42	8×42	8×42	8×42
4,4	239	239	42	42	1,0,0	70	35	70	35	0	0	50	15	50	15	0	0	0
10,10	490	490	6×41	6×41	2,0,0	112	42	112	42	0	0	42	7	42	7	0	0	0
1,0	70	35	50	15	10,0,0	422	69	422	69	0	0	8×39	8×3	8×39	8×3	0	0	0
2,0	112	42	42	7	0,1,0	54	88	20	20	68	34	34	68	0	0	68	34	34
3,0	152	47	40	5	0,2,0	60	130	20	20	110	40	6	42	0	0	42	6	6
4,0	192	50	40	3	0,10,0	87	438	20	20	422	67	8×3	8×39	0	0	8×39	8×3	8×3
10,0	422	67	6×38	6×3	1,1,0	103	103	70	35	70	35	83	83	50	15	70	35	35
0,1	54	88	34	68	2,2,0	152	151	111	41	111	41	49	48	41	6	41	6	6
0,2	60	130	6	42	10,10,0	489	488	423	68	422	68	8×42	8×42	8×39	8×3	8×39	8×3	8×3
0,3	65	168	5	38	0,1,1	54	88	54	88	101	101	34	68	34	68	101	101	101
0,4	68	210	3	42	0,2,2	60	129	60	129	150	150	6	41	6	41	49	49	49
0,10	87	437	6×3	6×38	0,10,10	87	439	87	439	486	485	8×3	8×39	8×3	8×39	8×42	8×42	8×42

On the other hand, Table 2 presents the volume of the *Ctr-Ctr* traffic (measured in kbps) between two or three Kandoo controllers. The right part of Table 2 gives the volume of the traffic exchanged between the root (or c_r) and one local (or c_l) controller, when various numbers of empty switches (no flows) are assigned to these controllers. The first column shows the number of switches at each controller. The next two columns show the traffic sent from one controller to the other, for both directions ($c_r \rightarrow c_l$ and $c_l \rightarrow c_r$), while the last two columns show the traffic increase when switches are added to the controllers. The increase at each row is relative to the previous row. For example, the last columns of the row starting with (1,1) show the 83 kbps traffic increase at each direction when an extra switch is assigned to each controller. The same columns of the row starting with (1, 0) indicate 50 kbps and 15 kbps traffic increases for $c_r \rightarrow c_l$ and $c_l \rightarrow c_r$ respectively, when one switch is added solely to c_r . Finally, the row starting with (0, 1) shows the corresponding 34 kbps and 68 kbps increases, when one

switch is added solely to c_l . We observe that as more and more switches are assigned to a controller, the increase of the traffic sent from this controller to the other one converges to 38 kbps per switch, while the corresponding increase in the opposite direction converges to 3 kbps per switch. Moreover, the traffic increase due to the assignment of extra x and y switches to the two controllers c_r and c_l respectively is approximately the sum of the individual traffic increases, which would happen if only x or y switches were assigned to only one of the two controllers c_r or c_l respectively. Based on these two observations, it is safe enough to assume that Remark 2 holds, and each extra switch increases the *Ctrl–Ctrl* traffic by $\beta^c \approx 38 + 3 = 41$ kbps.

The right part of Table 2 shows the corresponding traffic increases with three controllers, called root (or c_r), local-1 (or c_{l_1}) and local-2 (or c_{l_2}). Similarly, each extra switch assigned to a controller triggers an increase of its *Ctrl–Ctrl* traffic exchanged with other controllers, which is independent of the other assigned switches and converges to $\beta^c \approx 42$ kbps. Thus, Remark 2 is also confirmed by these results. Due to space limitations, we present a fraction of our experimental results, although we have been experimenting with more than three Kandoo controllers and obtaining the same qualitative conclusions. Finally, we observed that the increase of the *Ctrl–Ctrl* traffic, when flows are added to the switches, is minor and negligible.

7.2. The Heuristic Solution in the “Abilene” Topology

We use NITOS for deploying the network topology of “Abilene” from the Internet Topology Zoo Collection. We configure 11 NITOS raspberries to behave as OpenFlow switches, leveraging on OvS and three wireless interfaces. Each pair of nodes is connected with the use of a dedicated wireless frequency, which is not used by any other pair of nodes. Thus, there is no interference between the wireless links. For each pair, one node is the access point (AP) and the other one is the station, both in wireless distributed system (WDS) mode, which is required when the interfaces are added to OvS instances (they are not working in promiscuous mode). The total control traffic is measured with use of *nmetrics*. A special version of *nmetrics* is utilized, which is integrated with the OMF Monitoring Library (OML) [24] and enables the collection of the traffic measurements in a database, where OMF stands for the cOntrol and Management Framework for testbed experimentation. In addition, we measure the energy consumption for the transmission of the control traffic, using the Energy Monitoring Framework (EMF) [2] of NITOS. The wireless interfaces use the AR9380 chipset and the 802.11 g protocol, as well as specially designed devices capable of providing fined-grained measurements of the energy consumed by them. The hosts are virtually created at each node with the use of Mininet [25]. The flows of each switch are proactively configured and enable the hosts to ping each other.

In this experimentation, we configure each switch to have f flows, where $f = 250, 470$ or 750 . Using our previous results, $f\beta^s \approx 345$ kbps, 648.6 kbps or 1035 kbps and $\beta^c \approx 42$ kbps; thus, $f\beta^s / \beta^c \approx 8.21, 15.44$ or 24.64 . Figure 7 presents the optimal controller placement, as well as the placements returned by our base heuristic and by *local-search-variable* algorithm. For $f\beta^s / \beta^c = 8.21$, the optimal (Figure 7a) and heuristic (Figure 7d,g) solutions use the same number of controllers, equal to 3. The base heuristic includes “Houston” switch in its returned placement, because it features the third highest betweenness centrality. The optimal placement, however, includes a controller at “Denver” instead of “Houston,” despite its lower centrality metric. Due to this difference, the base heuristic solution requires approximately 6% more bandwidth than the optimal one. The *local-search-variable* algorithm is able to discover this improvement and applies it, returning the optimal placement.

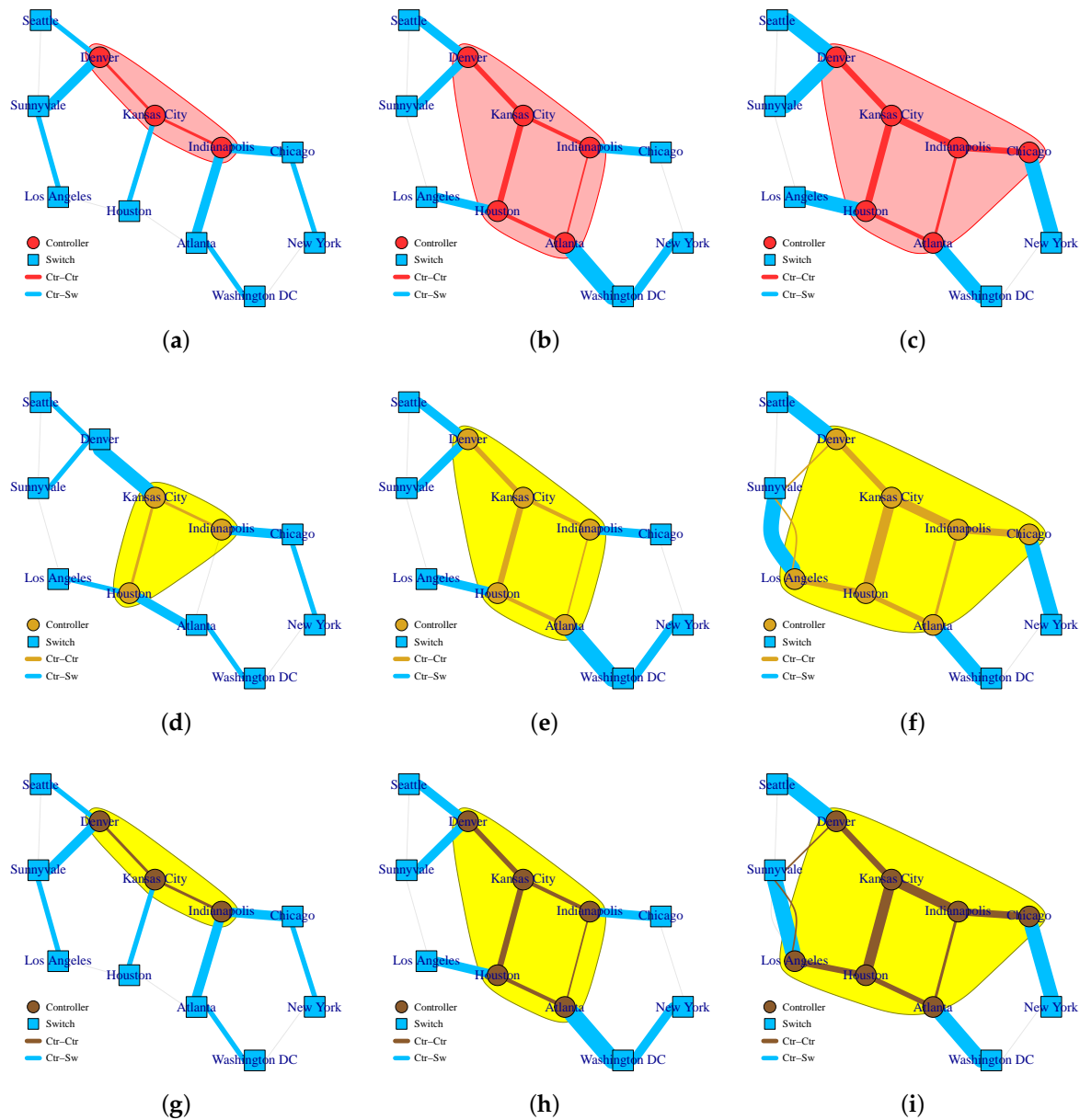


Figure 7. Visualization of the optimal and heuristic controller placements on the “Abilene” network topology for various $f\beta^s/\beta^c = 8.21, 15.44$ or 24.64 . The colour of each edge indicates whether the corresponding link is used for *Ctrl-Ctrl* and/or *Ctrl-Sw* traffic, while the edge width indicates the bandwidth used for this traffic. (a) Optimal placement ($f\beta^s/\beta^c = 8.21$). (b) Optimal placement for $f\beta^s/\beta^c = 15.44$. (c) Optimal placement for $f\beta^s/\beta^c = 24.64$. (d) Heuristic placement for $f\beta^s/\beta^c = 8.21$. (e) Heuristic placement for $f\beta^s/\beta^c = 15.44$. (f) Heuristic placement for $f\beta^s/\beta^c = 24.64$. (g) Local-search-variable placement for $f\beta^s/\beta^c = 8.21$. (h) Local-search-variable placement for $f\beta^s/\beta^c = 15.44$. (i) Local-search-variable placement for $f\beta^s/\beta^c = 24.64$.

For $f\beta^s/\beta^c = 15.44$, all three solutions are identical, as depicted in Figure 7b,e,h. Finally, for $f\beta^s/\beta^c = 24.64$, the heuristic solutions (Figure 7f,i) return a more extended set of controllers, compared to the optimal solution (Figure 7c). The result is an approximate 2% increase of the total required bandwidth for the control traffic.

In all cases of Figure 7, the energy consumption in the wireless interfaces, disabling non-control packets to be forwarded over these links, is proportionate to the total control traffic transmitted over

these links. These results prove that the minimization of the control traffic volume comes together with the minimization of the associated energy consumed.

8. Discussion and Future Work

In our search for efficient heuristics, we also experimented with some meta-heuristic approaches. One of them consisted of repeating the *local-search-fixed* algorithm multiple times, initiated by random initial placements. This is a typical approach to combat convergence of local search algorithms at local optima. Another meta-heuristic we examined is simulated annealing [26], which, at every step, decides whether to move to a neighbor solution chosen at random, with a probability that depends on the cost difference between the current and the neighbor solution and a control parameter called temperature which decreases with time. We have found both of these meta-heuristics to require substantially more time to reach the same performance as our proposed heuristics, which is explained by the fact that they do not use problem-specific information.

The selection of the betweenness centrality metric as a common element of our heuristic algorithms was made on the basis of its simplicity and the fact that it is calculated in very short time. As part of future work, we plan to extend our research by using other centrality metrics, apart from betweenness.

Even though the paper focused on the problem of minimizing the control traffic volume, the analysis can be easily extended with a bias for short paths between switches and controllers. Path hop counts are usually the major factor affecting latency in dense networks with negligible propagation delays. A goal of minimizing both the total control traffic and the average *Ctr-Sw* delay could be expressed as minimizing a weighted sum of the two quantities, which is the following

$$\begin{aligned} \arg \min_{\mathbf{c}} \left(\sum_{s \in \mathcal{S}} w^s f^s \beta^s + \sum_{(c_1, c_2) \in \mathcal{C}^2} w^{(c_1, c_2)} y^{c_1} \beta^c + \delta \sum_{s \in \mathcal{S}} w^s \right) = \\ \arg \min_{\mathbf{c}} \left(\sum_{s \in \mathcal{S}} w^s (f^s \beta^s + \delta) + \sum_{(c_1, c_2) \in \mathcal{C}^2} w^{(c_1, c_2)} y^{c_1} \beta^c \right), \end{aligned} \quad (15)$$

where δ is a parameter controlling the relative weight of the average *Ctr-Sw* hop count minimization in the objective. From the above formulation, it is evident that our results in this paper can be applied directly just by modifying the ratio $f\beta_s/\beta_c$ with $(f\beta_s + \delta)/\beta_c$.

9. Conclusions

In this work, we investigated the optimal controller placement and switch assignment to minimize total control traffic in a SDN-based IoT network, relying on *Ctr-Sw* and *Ctr-Ctr* traffic models experimentally validated in a testbed. We formulated the problem, which is NP-hard, using IQP, and proposed a set of heuristic algorithms that expedite the controller placement and switch assignment procedure, while incurring negligible traffic increases with respect to the optimal solution. Tested with a large number of topology graphs from the Internet Zoo Topology collection, the simplest of our heuristic solutions yielded approximately 4.5% more bandwidth than the optimal solution on average, while the most advanced yielded an average increase of approximately 1%, and notably, a worst case increase of less than 3%.

Author Contributions: Conceptualization, K.C.; software, K.C. and D.G.; writing—original draft preparation, K.C.; writing—review and editing, D.G.; supervision, P.F.; project administration, T.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research is co-financed by Greece and the European Union (European Social Fund-ESF) through the Operational Program “Human Resources Development, Education and Lifelong Learning 2014-2020” in the context of the project “Dynamically Reconfigurable Backhaul/Fronthaul in Software Defined 5G Networks” (MIS 5005859).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Choumas, K.; Giatsios, D.; Flegkas, P.; Korakis, T. The SDN Control Plane Challenge for Minimum Control traffic: Distributed or Centralized? In Proceedings of the CCNC, Las Vegas, NV, USA, 11–14 January 2019.
2. Keranidis, S.; Kazdaridis, G.; Passas, V.; Korakis, T.; Koutsopoulos, I.; Tassiulas, L. Online Energy Consumption Monitoring of Wireless Testbed Infrastructure Through the NITOS EMF Framework. In Proceedings of the 8th ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization, Miami, FL, USA, 30 September 2013.
3. Heller, B.; Sherwood, R.; McKeown, N. The Controller Placement Problem. In Proceedings of the HotSDN, Helsinki, Finland, 13 August 2012.
4. Nguyen, T.M.C.; Hoang, D.B.; Chaczko, Z. Can SDN Technology Be Transported to Software-Defined WSN/IoT? In Proceedings of the 2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Chengdu, China, 15–18 December 2016.
5. The Internet Topology Zoo. Available online: <http://www.topology-zoo.org/dataset.html> (accessed on 21 January 2020).
6. Ongaro, D.; Ousterhout, J. In Search of an Understandable Consensus Algorithm. In Proceedings of the USENIX ATC, Philadelphia, PA, USA, 19–20 June 2014.
7. The Raft Consensus Algorithm. Available online: <https://raft.github.io/> (accessed on 21 January 2020).
8. Zhang, T.; Bianco, A.; Giaccone, P. The role of inter-controller traffic in SDN controllers placement. In Proceedings of the IEEE Conference on NFV and SDN (NFV-SDN), Palo Alto, CA, USA, 7–9 November 2016.
9. OpenDaylight (ODL). Available online: <https://www.opendaylight.org/> (accessed on 21 January 2020).
10. Open Network Operating System (ONOS). Available online: <https://wiki.onosproject.org/> (accessed on 21 January 2020).
11. Lange, S.; Gebert, S.; Spoerhase, J.; Rygielski, P.; Zinner, T.; Kounev, S.; Tran-Gia, P. Specialized Heuristics for the Controller Placement Problem in Large Scale SDN Networks. In Proceedings of the International Teletraffic Congress (ITC), Ghent, Belgium, 8–10 September 2015.
12. Killi, B.P.R.; Rao, S.V. Capacitated Next Controller Placement in Software Defined Networks. *IEEE Trans. Netw. Serv. Manag.* **2017**, *14*, 514–527. [CrossRef]
13. Bari, M.F.; Roy, A.R.; Chowdhury, S.R.; Zhang, Q.; Zhani, M.F.; Ahmed, R.; Boutaba, R. Dynamic Controller Provisioning in Software Defined Networks. In Proceedings of the International Conference on Network and Service Management (CNSM), Zurich, Switzerland, 14–18 October 2013.
14. Ksentini, A.; Bagaa, M.; Taleb, T. On using SDN in 5G: The controller placement problem. In Proceedings of the IEEE GLOBECOM, Washington, DC, USA, 4–8 December 2016.
15. Mostafaei, H.; Menth, M.; Obaidat, M.S. A Learning Automaton-Based Controller Placement Algorithm for Software-Defined Networks. In Proceedings of the GLOBECOM, Abu Dhabi, UAE, 9–13 December 2018.
16. Su, Z.; Hamdi, M. MDCP: Measurement-aware distributed controller placement for software defined networks. In Proceedings of the IEEE ICPADS, Melbourne, Australia, 14–17 December 2015.
17. Qin, Q.; Poularakis, K.; Iosifidis, G.; Tassiulas, L. SDN Controller Placement at the Edge: Optimizing Delay and Overheads. In Proceedings of the IEEE INFOCOM, Honolulu, HI, USA, 16–19 April 2018.
18. Newman, M.E.J. *Networks: An Introduction*; Oxford University Press: Oxford, UK, 2010.
19. Bhardwaj, S.; Niyogi, R.; Milani, A. Performance Analysis of an Algorithm for Computation of Betweenness Centrality. In Proceedings of the ICCSA, Santander, Spain, 20–23 June 2011.
20. The R Project for Statistical Computing. Available online: <http://www.r-project.org> (accessed on 21 January 2020).
21. IBM ILOG CPLEX for Faculty. Available online: <http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer> (accessed on 21 January 2020).
22. Network Implementation Testbed Using Open Source Platforms (NITOS). Available online: <https://nitlab.inf.uth.gr/NITlab/nitos> (accessed on 21 January 2020).
23. Hassas Yeganeh, S.; Ganjali, Y. Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications. In Proceedings of the HotSDN, Helsinki, Finland, 13 August 2012.

24. Singh, M.; Ott, M.; Seskar, I.; Kamat, P. ORBIT Measurements framework and library (OML): Motivations, implementation and features. In Proceedings of the TRIDENTCOM, Trento, Italy, 23–25 February 2005.
25. Mininet: An Instant Virtual Network on Your Laptop. Available online: <http://mininet.org> (accessed on 21 January 2020).
26. Aarts, E.; Korst, J. *Simulated Annealing and Boltzmann Machines*; John Wiley & Sons: Chichester, UK, 1988.



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).