Dynamic SDN Controller Load Balancing

Sufiev, Hadar; Haddad, Yoram; Barenboim, Leonid; Soler, José

Published in:
Future Internet

Link to article, DOI:
10.3390/11030075

Publication date:
2019

Document Version
Peer reviewed version

Dynamic SDN Controller Load Balancing

Hadar Sufiev, Yoram Haddad, *Senior Member, IEEE,* Leonid Barenboim, and Jose Soler, *Senior Member, IEEE*

Abstract

The software defined networking (SDN) paradigm separates the control plane from the data plane, where an SDN controller receives requests from its connected switches and manages the operation of the switches under its control. Reassignments between switches and their controllers are performed dynamically, in order to balance the load over SDN controllers. In order to perform load balancing most dynamic assignment solutions use a central element to gather information requests for reassignment of switches. Increasing the number of controllers causes a scalability problem, when one super controller is used for all controllers and gathers information from all switches. In a large network, the distances between the controllers is sometimes a constraint for assigning them switches. In this paper, a new approach is presented to solve the well-known load balancing problem in the SDN control plane. This approach implies less load on the central element and meeting the maximum distance constraint allowed between controllers. An architecture with two levels of load balancing is defined. At the top level, the main component called Super Controller, arranges the controllers in clusters, so that there is a balance between the loads of the clusters. At the bottom level, in each cluster there is a dedicate controller called Master Controller, which performs a reassignment of the switches in order to balance the loads between the controllers. We provide a two-phase algorithm, called, Dynamic Controllers Clustering algorithm, for the top level of load balancing operation. The load balancing operation takes place at regular intervals. The length of the cycle in which the operation is performed can be shorter, since the top-level operation can run independently of the bottom level operation. Shortening cycle time allows for more accurate results of load balancing. Theoretical analysis demonstrates that our algorithm provides a near-optimal solution. Simulation results show that our dynamic clustering improves fixed clustering by a multiplicative factor of 5.

Index Terms

Multi Controllers; Architecture; SDN; Load Balancing

I. Introduction

In a software defined network (SDN) architecture [1] the logical separation between control plane and data plane, in the architecture and functional behavior of network nodes, is dissociated, allowing for centralization of all the logic related to control plane procedures in a so-called SDN Controller. In turn, this allows for simplified network nodes designed and streamlined for data plane performance. Such an architecture enables developers to devise new algorithms to be collocated at the SDN controller, which are able to manage the network and change its functionality [2], [3]. Even though only one controller may handle the traffic for a small network [4], this is not realistic when we deal with large network at the internet scale since each controller has limited processing power, therefore multiple controllers are required to respond to all requests on large networks [5], [6], [7]. One approach to achieve this goal is to use multiple same instantiations of a single controller [8], where each instantiation of the SDN controller does basically the same work as the others, each switch being linked to one SDN controller. Handling multiple controllers gave rise to some important question namely: where to place the controllers and how to match each control to a switch .These questions are not only relevant during the network deployment based on static's information [9], [10] but should also been considered regularly due to the network dynamic nature [11]. In this paper we dealt with the matching problem aforementioned. We proposed a novel muli-tier architecture for SDN control plane which can adapt itself to dynamic traffic load and therefore provides a dynamic load balancing.

Hadar Sufiev and Yoram Haddad are with the Department of Computer Science, Jerusalem College of Technology, Jerusalem, Israel e-mail: haddad@jct.ac.il; Leonid Barenboim is with The Open University of Israel, Raanana, Israel and Jose Soler is with Technical University of Denmark (DTU), Lyngby, Denmark

The rest of this paper is organized as follows. In section II we present the state of the art in dealing with load balancing at the control plane level in SDN. Then in section III we present the updated DCF architecture used to develop our clustering algorithm. In section IV, the problem is formulated and its hardness explained. The Two-phase DCC algorithm, with the running time analysis and optimality analysis are provided in section V. Simulations results are discussed in Section VI. Finally, our conclusions are provided in section VII.

II. Related Works

The problems of how to provide enough controllers to satisfy the traffic demand, and where to place them, were studied in [12], [13], [4]. The controllers can be organized hierarchically, where each controller has its own network sections that determines the flows it can serve [14], [15], [16], or in a flat manner where each controller can serve all types of incoming requests [17], [18], [19]. In any case every switch needs a primary controller (it can as well have more, as secondary/redundant). In most network $N >> M$, where N is the number of switches and M is the number of controllers, Therefore, each controller has a set of switches that are linked to it. The dynamic requests rate from switches can create a bottleneck at some controllers because each controller has limited processing capabilities. Therefore if the number of switches request is too large, the requests will have to wait in the queue before being processed by the controller which will cause long response times for the switches. To prevent the aforementioned issue, switches are dynamically reassigned to controllers according to their request rates [18], [20], [21]. This achieves a balance between the loads that the controllers have.
In general, these load-balancing methods split the timeline into multiple time slots (TSs) in which the load balancing methods are executed. At the beginning of each TS, these methods propose to run a load balancing algorithm based on the input gathered in the previous TS. Therefore, these methods assume the input is also relevant for the current TS. The load-balancing algorithm is executed by a central element called the Super Controller (SC). Some of the methods presented in the literature are adapted to dynamic traffics [19], [18]. They suggest changing the number of controllers and their locations, turning them on and off, in each cycle according to the dynamic traffic. In addition to load balancing, some other methods [6], [18] deal with additional objectives such as minimal set-up time and maximal utilization, which indirectly help to balance loads between controllers.[1] Changing the controller location causes reassignment of all its switches, thus, such approaches are designed for networks where time complexity is not a critical issue.

However, in our work, we do consider time sensitive networks, and therefore, we adopted a different approach that causes less noise in the network, whereby the controllers remain fixed and the reassignment of switches is performed only if necessary in the ongoing TS (as detailed further in this paper).
In [18], [20], [21] the SC run the algorithm that reassigns switches according to the dynamic information (e.g. switch requests per second) it gathers each time cycle (finding the optimal time cycle duration is the goal of our future works and is not considered in this paper) from all controllers, and changes the default controllers of switches. Note that each controller should publish its load information periodically to allow SC to partition the loads properly.
In [20] a load balancing strategy called "Balance flow" focuses on controller load balancing such that (1) the flow-requests are dynamically distributed among controllers to achieve quick response, (2) the load on an overloaded controller is automatically transferred to appropriate low-loaded controllers to maximize controller utilization. This approach requires each switch to enable to get service from some controllers for different flow. The accuracy of the algorithm is achieved by splitting the switch load between some controllers according the source and destination of each flow.
DCP-GK and DCP-SA, are greedy algorithms for Dynamic Controller Placement (DCP) from [18], which employ for the reassignment phase, Greedy Knapsack (GK) and Simulated Annealing (SA) respectively,

[1] The reassignment protocol between switches and controllers is out of the scope of this paper. Here we focus on the optimization and algorithmic aspects of the reassignment process. More details on reassignment protocol can be found in [22]. For instance, switch migration protocol [22] is used for enabling such load shifting between controllers and conforms with the Openflow standard. The reassignment has no impact on flow table entries.

Approach	Balance Flow	SMT	DCP-assignment phase	Hybrid flow
Time complexity	$O(max((N^2) \cdot log(N^2), N^2 \cdot M))$	$O(M \cdot N \cdot logN)$	$O(M \cdot N \cdot logN)$	$O(N \cdot M^2)$

TABLE I: Time complexity comparison

dynamically changing the number of controllers and their locations under different conditions, then, reassign switches to controllers.

Contrary to the methods in [18], [20], the algorithm suggested by [21] , called Switches Matching Transfer (SMT) , takes into account the overhead derived from the switch-to-controller and controller-to-switch messages. This method achieves good results as shown in [21].

In the approaches mentioned earlier in this section, all the balancing work is performed in the SC, thus, the load on the super controller can cause a scalability problem. This motivated the architecture defined in [23] called “Hybrid Flow”, where controllers are grouped into fixed clusters. In order to reduce the load on the SC, the reassignment process is performed by the controllers in each cluster, where the SC is used only to gather load data and send it to/from the controllers. “Hybrid Flow” suffers from long run time caused by the dependency that exists between the SC operation and other controllers operations.

When the number of controllers or switches increases, the time required for the balancing operation increases as well. Table I summarizes the time complexity of the methods mention above.

The running time of the central element algorithm defines the bound on the time-cycle length. Thus, the bigger the increase of the run time in the central element (i.e., causing a larger time cycle), the lower is the accuracy achieved in the load balancing operation. This is crucial in dynamic networks that need to react to frequent changes in loads [24].

In a previous work [25], [26] ,a new architecture called Dynamic Cluster Flow (DCF) [25] was presented. DCF facilitates a decrease in the running time of the balance algorithm. The DCF architecture partitions controllers into clusters. The architecture defines two levels of load balancing: A high level called ”*Clustering*”, and an operational level called ”*Reassignment*”. A super controller performs the ”*Clustering*”, by re-clustering the controllers in order to balance their global loads. The ”*Reassignment*” level is under the responsibility of each cluster that balances the load between its controllers by reassigning switches according to their request rate. For communication between the two levels, each controller has a Cluster Vector (CV), which contains the addresses of all the controllers in its cluster. This CV, which is updated by the SC each time cycle, allows the two levels to run in different independent elements, where the ”*Clustering*” operation runs at start time of each unit.

In [26] we presented a heuristic for the ”*Clustering*” operation which balances between clusters according to the loads with a time complexity of $O(M^2)$, and suggest to use the method presented in [23] for the ”*Reassignment*” level. In this initial architecture, the ”*Reassignment*” level is not sufficiently flexible for various algorithms, which served as a motivation for us to extend it.

In this paper, the target is to leverage on previous work [25], [26] and achieve load balancing among controllers. This is done by taking into account: network scalability, algorithm flexibility, minor complexity, better optimization and overhead reduction. To achieve the above objectives, we use the DCF architecture, and considered distance and load at the ”*Clustering*” level that influence the overhead and response time at the ”*Reassignment*” level, respectively.

Towards that target, the DCF architecture has been updated to enable the application of existing algorithms [18], [20], [21], [23] in the ”*Reassignment*” process in each cluster. Furthermore a clustering algorithm has been developed that takes into account the controller-to-controller distances in the load balancing operation. The problem has been formulated as an optimization problem, aiming at minimizing the difference between cluster loads with constraints on the controller-to-controller distance. This is a challenging problem due to these opposite objectives. We assume that each controller has the same limit capacity in terms of requests per second that it can manage. The controllers are dynamically mapped to clusters when traffic changes. The challenge is to develop an efficient algorithm for the mentioned problem i.e. re-clustering in response to variations of network conditions, even in large-scale SDN networks. We propose a novel **D**ynamic **C**ontrollers **C**lustering (further denoted as DCC) algorithm by defining our problem as a K-Center problem at the first phase and developing, in a second phase, a replacement rule to swap controllers between clusters. The idea for the second phase is inspired by Game Theory. The replacements shrink the gap between cluster loads while not exceeding the constraint of controller-to-controller distances within the cluster. We assume that M controllers are sufficient for handling the maximal request rate in the network (as mentioned earlier, there are already many works that found the optimal number of controllers).

Our architecture and model are different from aforementioned existing works since we enable not only dynamicity inside the cluster but also between clusters which didn't exist before.

III. Network Architecture

A. Dynamic Cluster Flow Architecture

In the architecture we presented in [25], [26] we considered always only one controller called Super Controller (SC) that is connected to all other controllers in the control plane. In this sense, we considered a two-tier hierarchy where the SC is in the top tier and all other controllers are in the lower tier. The SC gathers load information from all controllers and is responsible for grouping the controllers into clusters according to their load. For each cluster, the SC handles a Cluster Vector (CV), which includes the addresses of all the controllers inside the given cluster. The CV provides each controller inside a cluster the ability to identify to which controller it can transfer part of its overload. In order for each controller to intelligently decide where to transfer its load, an efficient communication scheme between the controllers is necessary. Because each overloaded controller that requires the help of other controller becomes dependent on the other controller which itself can be also overloaded etc. This whole interdependency requires a complete and synchronized coordination between all the controllers.

B. Three level load balancing architecture

In this paper, we consider the architecture, presented in [25], [26] as briefly presented in the previous section but we add a new additional intermediate tier.

Thus we consider a three-tier hierarchy, where the SC is in the top tier, some Master Controllers in the intermediate tier and all other controllers are in the lower tier. In figure 1 we show an illustration of our proposed architecture with an example of reclustering process where in green we see the master of each cluster and SC denotes the super controller. In this three-tier architecture, we observe two levels of load operations: "*Clustering*" and "*Reassignments*". In the top level of "*Clustering*", the SC organizes the controllers into clusters. In the lower level of "*Reassignments*", for each cluster, there is a Master Controller (*MC*) responsible for the load balancing inside the cluster by reassigned switches to controllers dynamically. In order to define the problem of the "*Clustering*" for the high level of the load balancing operation, two aspects are considered: first, the minimal differences between clusters' loads are set as targets, and second, the minimal distances between controllers in each cluster.

In this paper, we do not propose a new method for performing load balancing inside each cluster since many efficient algorithms already exist, for instance see [18], [20], [21]. However, the method employed, which is designed for the top level in which clusters are rearranged (as described in the next section), is sufficiently generic that it will operate with any of the existing load-balancing algorithms.

At the start of each time cycle, the MC sends to the SC the Cluster Vector Loads (CVL), which includes the load of each controller inside the cluster. Then the SC may update the CV of each cluster in order to balance the load of some overloaded clusters, and may also update who is the new the new MC of some cluster. In parallel to the "*Clustering*" operation performed by the SC, the "*Reassignments*" operations of the load balancing are performed by the MS independently.

Due to the three level DCF architecture, the load balancing runtime of both the SC and MC is very low, and enables a reduction in the TS duration accordingly. Thus the greater the reduction in the timeline, the greater the accuracy achieved [25].

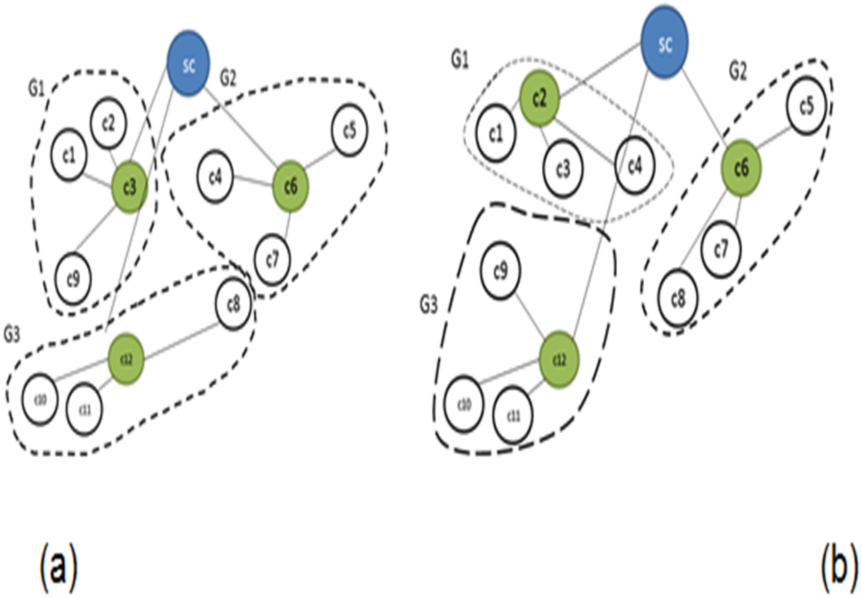

Fig. 1: Three-tier control plane

IV. DCC Problem Formulation

In this section, the assignment problem of controllers to clusters is presented. This problem is considered here as a minimization problem with constraints.

A. Notations

We consider a control plane C with M controllers, denoted by $C = \{C_1, C_2, ..., \ C_M\}$ where C_i is a single controller. We assume that the processing power of each controller is the same and equal to P, which stands for the number of requests per second that it can handle. Let d_{ij} be the distance (number of hops) between C_i and C_j. We denote by G_i the $i_t h$ cluster and by $G = \{G_1, G_2, ..., \ G_K\}$, the set of all clusters. We assume that $\frac{M}{K}$ is an integer and is actually the number of controllers per cluster. Thus, the size of the CV is $\frac{M}{K}$, i.e. we assume that each cluster consist of the same number of controllers. Y denotes a matrix, handled by SC, which consists of the matching of each controller to a single cluster. Each column of Y represents a cluster and each row a controller.

Figure 2 shows an example of such a matrix $Y(9X3)$ corresponding to nine controllers split into three clusters. On the left part we see Y matrix before re-clustering process and on the right after it is completed. Thanks to Y one can know which controller is in which cluster as follows. If a controller is included in a cluster then in the corresponding row of the controller and corresponding column of the cluster the value will be 1 otherwise it will be 0. For instance, we see that before re-clustering controller number 5 is in cluster b. After re-clustering (on the right matrix) we see that controller 5 is no longer in b but has been moved to cluster c.

Therefore, Y is a binary MXK matrix as follows:

$$Y(t)_{ij} = \left\{ \begin{array}{cc} 0 & C_j \in G_i \\ 1 & \text{else} \end{array} \right\} \tag{1}$$

$$\forall_{1 \le j \le M} \quad \sum_{i=1}^{K} Y(t)_{ij} = 1 \quad and \quad \forall_{1 \le j \le K} \quad \sum_{i=1}^{M} Y(t)_{ij} = K \tag{2}$$

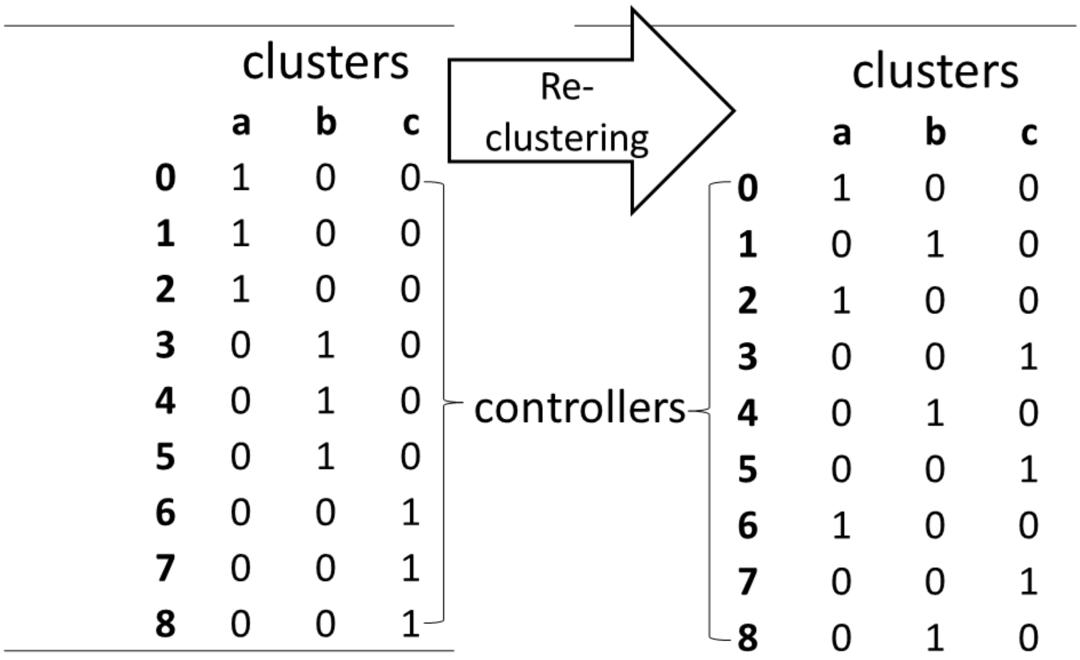

Fig. 2: Y matrix examples before and after re-clustering

Symbol	Semantics
C_j	j_{th} controller
G_i	i_{th} cluster
P	the number of requests a controller can handle per second
d_{ij}	Minimal hop distance between C_i and C_j
$Y(t)_{ji}$	$Y(t)_{ji} = 1$ if j_{th} controller is in cluster i in time slot t, else $Y(t)_{ji} = 1$
$l(t)_j$	Controller load - Average flow request of j_{th} controller per second in time slot t
SC	Super Controller - collects controllers' loads from masters and re-clustering

TABLE II: Key notations

The load of controller j in time slot t is denoted $l(t)_j$. This information arrives to the SC from the controllers. Its value is the average of requests per second from all the switches associated to the controller, in time slot t. CVL_i denotes the Cluster Vector Load of $Master_i$. The SC contains the addresses of the masters for each cycle in the Master Vector (MV). Table II summarizes the key notations for ease of reference.

B. Clusters' Load Differences

As we mentioned in section III, the first aspect of the high-level load balancing is to achieve balanced clusters (in this paper we assume that all controllers have the same processing capabilities, therefore balanced clusters is the overall optimal allocation). For this purpose, the gaps between their loads must be narrowed. A cluster load is defined as the sum of the controllers' average loads included in it, as follows:

$$\theta(t)_i = \sum_{j=1}^{M} l(t)_j \, Y(t)_{ji} \tag{3}$$

Where i is the cluster number and M is the number of controllers in the cluster.
To measure how much a cluster load is far from other clusters' loads, we derive the global cluster's load average:

$$Avg = \frac{\sum_j^M l(t)_j}{K} \tag{4}$$

Where, k is the number of clusters.
Then, we define the distance of a cluster's load from the global average's load (denoted above Avg) as:

$$\vartheta(t)_i = |\theta(t)_i - Avg| \tag{5}$$

In a second step, we define a metric that measures the total load difference between clusters' load as follows:

$$\varsigma(t) = \frac{\sum_{i=1}^k \vartheta(t)_i}{K} \tag{6}$$

C. Distances between controllers within same cluster

In this section, we focus on the second aspect of the high level load balancing (mentioned in section III-B). The rationale behind this distance optimization is as follows. Since in the initialization phase, switches are matched to the closest controller, then when we perform load balancing inside the same cluster we want the other controllers to be also close to each other, otherwise this would imply that the switch might be now matched to a new controller far from it. For that purpose we define the maximal distance between controllers within the same cluster (over all the clusters) as follows:

$$\eta(t) = max_{1\le c\le K} \quad max_{1\le i,j\le M} \quad d_{ij} Y(t)_{ic} \, Y(t)_{jc} \tag{7}$$

Where c is the cluster number, and i, j are the controllers in cluster c.
Obviously, the best result would be to reach the minimum $n(t)$ possible. Because if the controllers are close to each other, then the overhead consisting of the message exchanged between them will be less significant whereas if they are far from each other, then a multihop path will be required which will clearly impact the traffic on the control plane. However if the constraint on $n(t)$ is too strict this might not allow us enough flexibility to perform load balancing. Therefore we propose to define the minimum distance required to provide enough flexibility for the load balancing operation, denoted as "*minMaxDistance*". If the value of *minMaxDistance* is not large enough, it is possible to adjust it by adding an offset to it. Finally we denote by *Cnt* the constraint on the maximal distance as follows:

$$Cnt = maxDistance + \text{offset} \tag{8}$$

D. Optimization Problem: Dynamic Controllers' Clustering

Our goal is to find the best clustering assignment as defined by $Y(t)$ which minimize $\varsigma(t)$ (Eq.6) and at the same time fulfills the distance constraint (Eq. 8). Therefore, the problem can be formulated as follows:

$$Minimize \quad \varsigma(t) \tag{9}$$

subject to:

$$\sum_{j}^{M} Y(t)_{ji} = \frac{M}{K} \quad , \quad \forall_i \tag{10}$$

$$\sum_{j}^{K} Y(t)_{ji} = 1 \quad , \quad \forall_j \tag{11}$$

$$\eta(t) < Cnt \tag{12}$$

$$Y(t)_{ij} \in 0, 1 \quad , \quad \forall_{i,j}$$

Equation 10 ensures that each cluster has exactly M/K controllers at a given time while Equation 11 ensures that each controller is assigned to exactly one cluster at a time. Equation 12 puts a constraint on the maximum distance between controllers within same cluster.
Regarding the distance constraint, the problem is a variant of a k-Center problem [27].

On the other hand, the load balancing problem is a variant of a coalition-formation game problem [28], where the network structure and the cost of cooperation play major roles.

These two general problems are NP-Complete because finding an optimal partition requires iterating over all the partitions of the player set, where the number of these partitions grows exponentially with the number of players, and is given by a value known as the Bell Number [29]. Hence, finding an optimal partition in general is computationally intractable and impractical (unless $P = NP$).
In this paper, we propose an approximation algorithm to solve these problems. We adapt the K-Center problem solution for initial clustering, and use game theoretic techniques to satisfy our objective function with the distance constraint.

V. DCC Two Phase Algorithm

In this section, we divide the DCC problem into two phases and present our solutions for each of them. In the first phase, we define the initial clusters. We show some possibilities for the initialization that refer to distances between controllers and load differences between clusters. In the second phase, we improve the results. We further reduce the differences of cluster loads without violating the distance constraint by means of our replacement algorithm. We also discuss the connections between these two phases, and the advantages of using this two-phase approach for optimizing the overall performance.

A. Phase 1: Initial Clustering

The aim of initial clustering is to enable the best start that provides the best result for the second phase. There are two possibilities for the initialization. The first possibility is to focus on the distance, that is, seeking an initial clustering which satisfy the distance constraint while the second possibility is to focus on minimizing load difference between clusters.

1) Initial clustering with the distance constraint: Most of the control messages concerning the cluster load balancing operation are generated due to the communication between the controllers and their MC. Thus, we use the K-Center problem solution to find the closer MC [30], [27]. In this problem, $C = \{C_1, C_2, ..., \ C_K\}$ is the center's set and $P = \{P_1, P_2, ..., \ P_M\}$ contains M controllers. We define $P_C = (d(p_1, C), d(p_2, C), \ldots, d(p_M, C))$, where the i^{th} coordinate of P_C is the distance of p_i to its closest center in C. The k-Center inputs are: a set P of M points and an integer number K, where $M \in N$, $K < M$. The goal is to find a set of k points $C \subseteq P$ such that the maximum distance between a point in P and its closest point in C is minimized. The network is a complete graph, and the distance definition [see Table II] satisfies the triangle inequality. Thus, we can use an approximate solution to the k-Center problem to find MCs. Given a set of centers, C, the k-center clustering price of P by C is $\|P_C\|_\infty = max_{p \in P} d(p, C)$. Algorithm 1 is an algorithm similar to the one used in [31]. This algorithm computes a set of k centers, with a 2-approximation to the optimal k-center clustering of P, i.e., $\|P_K\|_\infty \leq 2opt_\infty(P, K)$ with $O(MK)$ time and $O(M)$ space complexity [31].

Algorithm 1 Find masters by 2-approximation greedy k-center solution

---

input: $P = \{p_1, p_2, ..., p_M\}$ controllers set, controller-to-controller matrix distances
output: $C = \{c_1, c_2, ..., c_K\}$ masters set
procedure:

1: $C \leftarrow \emptyset$
2: $c_1 \leftarrow p_i$ // an arbitrary point p_i from
3: $C \leftarrow C \cup c_1$
4: **for** $i = 1 : K$ **do**
5:     **for all** $p \in P$ **do**
6:         $d_i[p] \leftarrow min_{c \in C} \quad d(p, c)$
7:     **end for**
8:     $c_i \leftarrow max_{p \in P} \quad d_i[p]$
9:     $C \leftarrow C \cup c_i$
10: **end for**
11: **return** C // The master set

In Line 2 the algorithm chooses a random controller as the first master. In Lines 4-6 the algorithm computes the distances of all other controllers from the masters chosen in previous iteration. In each iteration,in line 9, another master is added to the collection, after calculating the controller located in the farthest radius of all controllers already included in the master group, in line 8. After $(K - 1)$ iterations in line 11 the set of masters is ready.

After that Algorithm 1 finds K masters, we partition controllers between the masters by keeping the number of controllers in each group under M/K as illustrated in Heuristic 2.

As depicted in Heuristic 2, lines 1-2 prepare set S that contains the list of controllers to assign. lines 3-5 define the initial empty clusters with one master for each one. Lines 7-15 (while loop) are the candidate clusters which have less than M/K controllers, and each controller is assigned to the nearest master of these candidates. After the controllers are organized into clusters, we check the maximal distance between any two controllers in lines 16-19; this value is used for the "*maxDistance*" (that was used for Eq. 8).

Heuristic 2 Distance initialization

input:

$C = \{c_1, c_2, ..., c_M\}$ Controller list

$M = \{m_1, m_2, ..., m_k\}$ masters list

controller-to-controller matrix distances

output:

$CL = \{cl_1, cl_2, ..., cl_k\}$ Clusters list, where $CL_i = \{cl_{i1}, cl_{i2}, ..., cl_{i(m/k)}\}$

maxDistance - maximum distance between controllers in a cluster.

procedure:

1: $S \leftarrow C$
2: $S \leftarrow S - M$
3: **for** $i = 1 : K$ **do**
4: $\quad CL_i \leftarrow M_i$
5: **end for**
6: $Candidates \leftarrow CL$
7: **while** $S \neq \emptyset$ **do**
8: $\quad C_{next} \leftarrow$ The next controller in S
9: $\quad CL_{near} \leftarrow$ Find the nearest master from Candidates list
10: $\quad CL_{near} \leftarrow CL_{next} \cup CL_{near}$
11: $\quad S \leftarrow S - C_{next}$
12: $\quad$ **if** $| CL_{near} |= M/K$ **then**
13: $\qquad Candidates \leftarrow Candidates - CL_{near}$
14: $\quad$ **end if**
15: **end while**
16: **for all** $CL_i \in CL$ **do**
17: $\quad maxDistanceCL_i \leftarrow$ max distance between two controllers in CL_i
18: **end for**
19: $maxDistance \leftarrow$ maximum of all $maxDistanceCL_i$
20: **return** $CL, maxDistance$

Regarding the time complexity, Lines 1-6 take $O(K)$ time. For each controller Line 8-12 checks the distance of a controller from all candidates, which takes $O(M * K)$ time. In lines 16-18 for each cluster the heuristic checks the maximum distance between any two controllers in the cluster.There are M/K^2 different distances for all clusters, thus taking $O(M^2)$ time. Line 19 takes $O(K)$ time ($K < M$). The initial process with Heuristic 2 entails an $O(M^2)$ time complexity.

Heuristic 2 is based on the distances between the controllers. When the controllers' position is fixed, the distances do not change. Consequently, heuristic 2 can be calculated only one time (i.e., before the first cycle) and the results are used for the remaining cycles.

2) Initial clustering based on load only: If the overhead generated by additional traffic to distant controllers is not an issue (for example due to broadband link) then we should consider this type of initialization, which put an emphasis on the controllers' load. In this case, we must arrange the controllers into clusters according to their loads. To achieve a well-distributed load for all the clusters we want to reach a "$min - max$", i.e., we would like to minimize the load in the most loaded cluster. As mentioned earlier (in IV-A) we assume the same number of controllers in each cluster. We enforce this via a constraint on the size of each cluster (see further Heuristic 3).

In the following, we present a greedy technique to partition the controllers into clusters (Heuristic 3). The basic idea is that in each iteration it fills the less loaded clusters with the most loaded controller.

In Heuristic 3, line 1 sorts the controllers by loads. In Line 2-9, each controller, starting with the heaviest one, is matched to the group with the minimum cost function, $Cost_g(C)$, if the group size is less than K, where

$$Cost_g(C) = CurrentClusterSum + C_{load}. \tag{13}$$

The "*CurrentClusterSum*" is the sum of the controllers' loads already handled by cluster g, and C_{load} is the controller's load that will be handled by that cluster. Regarding the time complexity, sorting M controllers takes $O(M \cdot log_2 M)$ time. Adding each controller to the current smallest group takes $M \cdot K$ operations. Therefore, heuristic 3 has $O(max(M \cdot K, M \cdot lgM))$ time complexity.

Heuristic 3 Load initialization

input:
$C = \{c_1, c_2, ..., c_M\}$ Controller list
Masters CVL_i's (average flow-request number (loads) for each controller)
integer K for number of clusters
output:
$P = \{p_1, p_2, ..., p_K\}$ Clusters list, where $P_i = \{c_{i1}, c_{i2}, ..., c_{i(M/K)}\}$
procedure:

```
SortedListC ← descending order of controllers list according to their loads
Candidates ← P
for all c ∈ SortedListC do
    P_min ← the cluster with minimal Cost_g(C) from candidates
    P_min ← P_m in cupC
    if | P_min |= M/K then
        Candidates ← Candidates − P_min
    end if
end for
return P
```

B. Initial Clustering as Input to the Second Phase

The outcomes of the two types of initialization, namely "*distance*" and "*load*", presented so far (section V-A) are used as an input for the second phase.
It should be noted that since the "*maxDistance*" constraint is an output of the initialization based on the distance (Heuristic 2), the first phase is mandatory in case the distance constraint is tight. On the other hand, the initialization based on the load (Heuristic 3) is not essential to being perform load balancing in the second phase, but it can accelerate convergence in the second phase.

C. Phase 2: Decreasing Load Differences using a Replacement Rule

In the second phase, we apply the coalition game theory [28]. We can define a rule to transfer participants from one coalition to another. The outcome of the initial clustering process is a partition denoted Θ defined on a set C that divides C into K clusters with M/K controllers for each cluster. Each controller is associated with one cluster. Hence, the controllers that are connected to the same cluster can be considered participants in a given coalition.
We now leverage the coalition game-theory in order to minimize the load differences between clusters or to improve it if an initial load balancing clustering has been performed such as in V-A2
A coalition structure is defined by a sequence $B = \{B_1, B_2, \ \ldots, \ B_l\}$ where each B_i is a coalition. In general, a coalition game is defined by the triplet (N, v, B), where v is a characteristic function, N are the elements to be grouped and B is a coalition structure that partitions the N elements [28]. In our problem the M controllers are the elements, G is the coalition structure, where each group of controllers G_i is a coalition. Therefore, in our problem we can define the coalition game by the triplet (M, v, G) where $v = \varsigma(t)$. The second phase can be considered as a coalition formation game. In a coalition formation game each element can change its coalition providing this can increase its benefit as we will define in the following. For this purpose, we define the Replacement Value (RV) as follows:

$$RV(C_i, C_j, a, b, Cnt) = \left\{ \begin{array}{cc} 0 & n(t)_{new} \geq Cnt \\ 0 & belowAverage \text{ is true} \\ 0 & aboveAverage \text{ is true} \\ sum_{new} - sum_{old} & \text{else} \end{array} \right\} \tag{14}$$

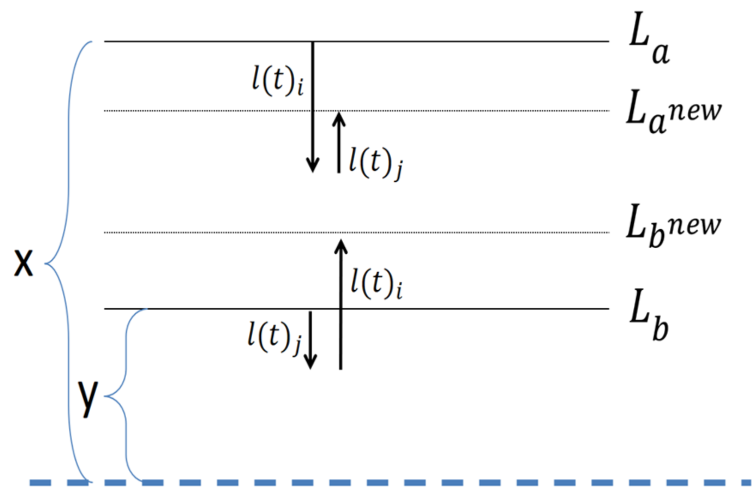

Fig. 3: Clusters loads after replacement on the same side with reference to the average

Where $sum_{new} = \vartheta(t)_{a^{new}} + \vartheta(t)_{b^{new}}$ and $sum_{old} = \vartheta(t)_{a^{old}} + \vartheta(t)_{b^{old}}$. $belowAverage$ is true where $(\vartheta(t)_{a^{old}} \leq Avg)\&(\vartheta(t)_{b^{old}} \leq Avg)$ and $aboveAverage$ is true where $(\vartheta(t)_{a^{old}} \geq Avg)\&(\vartheta(t)_{b^{old}} \geq Avg)$. Each replacement involves two controllers C_i and C_j with loads $Cl(t)_i$ and $l(t)_j$, respectively, and two clusters a and b with loads L_a and L_b, respectively. We use the notations "*old*" and "*new*" to indicate a value before and after the replacement.

When $n(t)_{new} \geq Cnt$(see Equations 7 and 8), the controllers, after the replacement, are organized into clusters such that the maximum distance between controllers within a particular cluster exceeds the distance constraint Cnt. In this case, the value of the RV is set to zero, because the replacement is not relevant at all.

When $(\vartheta(t)_{a^{old}} \leq Avg)\&(\vartheta(t)_{b^{old}} \leq Avg)$ or $(\vartheta(t)_{a^{old}} \geq Avg)\&(\vartheta(t)_{b^{old}} \geq Avg)$ (see Equations 4 and 5), $\varsigma(t)_{old} = \varsigma(t)_{new}$ (see Equation 6). When one of the cluster's load moves to another side of the global average then we have $\varsigma(t)_{new} \geq \varsigma(t)_{old}$. With both options, $\varsigma(t)_{old}$ do not improve and therefore the RV is set to zero.

Figure 3 and Figure 4 provide an illustration of these two options. The dotted line denotes the average of all clusters.

In Figure 3, the sum of the loads' distances from the global average, before the replacement is $x+y$. After the replacement the sum is $(x - (l(t)_i + l(t)_j) + y + (l(t)_i + l(t)_j) = x + y$. In the other symmetrical options, the result is the same.

In Figure 4 the sum of distances from the global average, before the replacement is $x + y$, and this sum after the replacement is $(x + (l(t)_i + l(t)_j) + (l(t)_i + l(t)_j) - y > x + y$. In the other symmetrical options, the result is the same.

In Equation 14, If none of the first three conditions are met, RV is calculated by $(\vartheta(t)_{a^{new}} + \vartheta(t)_{b^{new}}) - (\vartheta(t)_{a^{old}} + \vartheta(t)_{b^{ol}}$ a value that can be greater than or less than zero. Using the RV, we define the following "*ReplacementRule*":

Definition 1. Replacement Rule. In a partition Θ, a controller c_i has incentive to replace its coalition a with controller c_j from coalition b (forming the new coalitions $a^{new} = \left(a^{old} \backslash c_i\right) \cup c_j$ and $b^{new} = \left(b^{old} \backslash c_j\right) \cup c_i$ if it satisfies both of the following conditions: (1) The two clusters a^{new} and b^{new} that participate in the replacement do not exceed their capacity $K \cdot P$. (2) The RV satisfies: $RV\left(C_i, C_j, a, b, Cnt\right) < 0$ RV defined in Equation 14)

In order to minimize the load difference between the clusters we find iteratively a pair of controllers

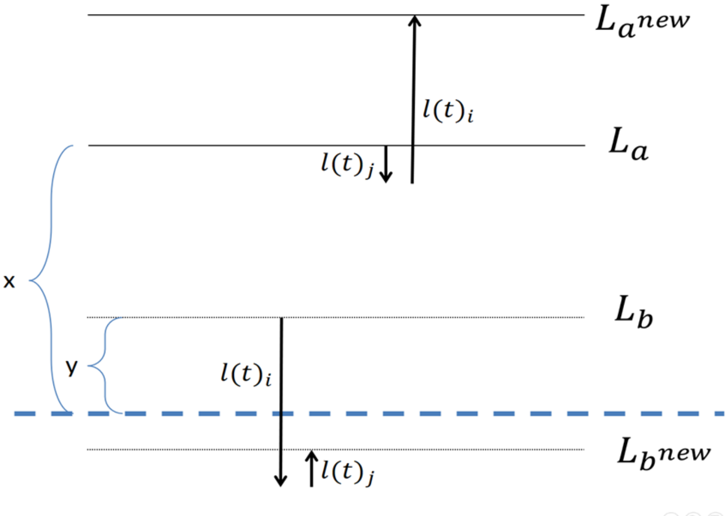

Fig. 4: Clusters' loads after replacement on different sides with reference to the average

with minimum RV, which then implement the corresponding replacement. This is repeated until all RV's are larger than or equal to zero: $\mathrm{RV}\big(C_i, C_j, a, b, Cnt\big) > 0$. Algorithm 4 describes the replacement procedure.

Regarding the time complexity of lines 1 in algorithm 4, i.e., find the best replacement, it takes: $\frac{M}{k}(k-1) + \frac{M}{k}(k-2) + ... + \frac{M}{k}(k-(k-1)) = \frac{M^2}{k^2} \cdot \frac{k(k-1)}{2} = \frac{M^2(k-1)}{2k} = o\big(M^2\big)$ time.

Line 3 invokes the replacement within $O(1)$ time. Since in each iteration the algorithm chooses the best solution, there will be a maximum of $M/2$ iterations in the loop of lines 2-5. Thus, in the worst case Algorithm 4 takes an $O(M^3)$ time complexity. In practice, the number of iterations is much smaller, as can be seen in the simulation section.

Procedure 4 Replacements

input:
$CL = \{cl_1, cl_2, ..., cl_K\}$ Clusters list, where $CL_i = \{cl_{i1}, cl_{i2}, ..., cl_{i(M/K)}\}$
distance constraint Cnt
output:
$CL = \{cl_1, cl_2, ..., cl_K\}$ Clusters list after replacements
procedure:
1: $bestVal \leftarrow \min_{c_i \in CL_x, c_j \in CL_y, x \neq y, 1 \leq x,y, \leq k} RV(c_i, c_j, CL_x, CL_y, Cnt)$
2: **while** $bestVal < 0$ **do**
3: invoke replacement RV
4: $bestVal \leftarrow \min_{c_i \in CL_x, c_j \in CL_y, x \neq y, 1 \leq x,y, \leq k} RV(c_i, c_j, CL_x, CL_y, Cnt)$
5: **end while**
6: **return** CL

D. Dynamic Controller Clustering Full Algorithm

Now we present the algorithm that includes the two stages of initialization and replacement, in order to obtain clusters in which the loads are balanced.

Algorithm 5 DCC Algorithm

input:
nt Network contain $C = \{c_1, c_2, ..., c_M\}$ Controller list, and distances between controllers.
K and M for the number of clusters and controllers, respectively
$constaintActive$ to indicate that it meets the controller-to-controller distance constraint
$offset$ to calculate the distance constraint (optional).
output:
$P = \{p_1, p_2, ..., p_k\}$ Clusters list, where $P_i = \{c_i1, c_i2, ..., c_i(m/k)\}$
procedure:

1: **if** $constaintActive = true$ **then**
2: $Masters \leftarrow$ Algorithm1(nt)
3: ($initialDistanceClusters$,$maxDistance$) $\leftarrow$ Heuristic2(C,$Masters$)
4: $Cnt \leftarrow maxDistance + offset$
5: $finalPartition \leftarrow$ Algorithm4($initialDistanceClusters$,$true$,Cnt)
6: **else**
7: $initialStructure \leftarrow$ Cluster structure from the previous cycle
8: $initialLoadsOnly \leftarrow$ Heuristic3(c)
9: $initialWithReplacement \leftarrow$ Algorithm4($initialLoadsOnly$,$false$)
10: $ReplacementOnly \leftarrow$ Algorithm4($initialStructure$,$false$)
11: $finalPartition \leftarrow$ best solution from($initialLoadsOnly$, $initialWithReplacement$, $ReplacementOnly$)
12: **end if**
13: **return** $finalPartition$

The DCC Algorithm runs the appropriate initial clustering, according to a Boolean flag called "*constraintActive*", indicating whether the distance between the controllers should be considered or not (Line 1). If the flag is true, the distance initialization procedure (Heuristic 2) is called (line 3). Using the "*maxDistance*" output from Heuristic 2, the DCC calculates the $Cnt = maxDistance + offset$ (Line 4). Using the partition and Cnt outputs, the DCC runs the "*replacementprocedure*" (Algorithm 4) (Line 5).

The DCC can run the second option without any distance constraint (Line 6). In Line 11 it chooses the best solution in such cases, (referring to the minimal load differences) from the following three options: (1) Partition by loads only (Line 8);(2) Start partition by loads and improve with replacements (Line 9) (3) Partition by replacements only (using the previous cycle partition) (Line 10).

Regarding the time complexity, DCC uses heuristic 2, heuristic 3, algorithm 1 and algorithm 4, thus it has a $O\left(M^3\right)$ time complexity.

E. Optimality Analysis

In this section, our aim is to prove how close our algorithm is to the optimum. Because the capacity of controllers is identical, the minimal difference between clusters is achieved when the controllers' loads are equally distributed among the clusters, where the clusters' loads are equal to the global average, namely $\varsigma(t) = 0$. Since in the second phase, i.e., in the replacements, the DCC full algorithm is the one that sets the final partition and therefore determines the optimality, it is enough to provide proof of this.
As mentioned before, the replacement process is finished when all $RV's$ are 0, at which time any replacement of any two controllers will not improve the result. Figure 5 shows the situation for each two clusters at the end of the algorithm.

For each two clusters, where the load of one cluster is above the general average and the load of the second cluster is below the general average, the following formula holds:

$$\vartheta(t)_a + \vartheta(t)_b = L_a - L_b \leq l(t)_i - l(t)_j, \forall c_i \in a, c_j \in b \quad (15)$$

We begin by considering the most loaded cluster and the most under-loaded cluster. When the cluster size is g, we define X_1 to contain the lowest $g/2$ controllers, and X_2 to contain the next lowest $g/2$ controllers. In the same way, we define Y_1 to contain the highest $g/2$ controllers and Y_2 to contain the next highest $g/2$ controllers. In the worst case, the upper cluster has the controllers from the Y_1 group and the lower cluster has the controllers from the X_1 group. Since the loads of the clusters are balanced,

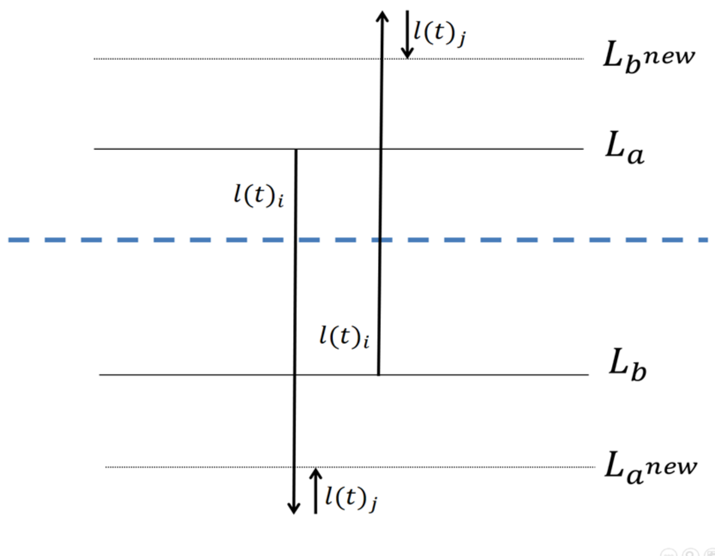

Fig. 5: The loads of each two clusters at the end of all replacements

Algorithm 1	Find Masters	$O(M \cdot K)$
Heuristic 2	Distance Initialization	$O(M^2)$
Heuristic 3	Load Initialization	$O(max(M \cdot \lg M, M \cdot K))$
Algorithm 4	Replacements	$O(M^2)$
Algorithm 5	General DCC	$O(M^3)$

TABLE III: Time Complexity of heuristics and algorithm used for DCC

one half of the controllers in the upper cluster are from X_1, and the other half of controllers in the lower cluster are controllers from Y_1.

According to Formula 15, we can take the lowest difference between a controller in the upper cluster and a controller in the lower cluster to obtain a bound on the sum of the distance of loads of these two clusters from the overall average. The sum of distances from the overall average of these two clusters is equal to or smaller than the difference between the two controllers, i.e., between the one with the lowest load of the g most loaded controller and the one with the highest load of the g lowest controllers.

$$\vartheta(t)_{most\_loaded} + \vartheta(t)_{most\_under\_loaded} \leq l(t)_{g\_th\_bigger} - l(t)_{g\_th\_smaller} \tag{16}$$

The bound derived in (Eq. 16) is for the two most distant clusters. Since a bound for the whole network (i.e. for all the clusters is needed) we just have to to multiply this bound by the number of clusters pairs we have in the network. There are k clusters in the networks so $k/2$ pairs of clusters, therefore the bound in (Eq. 16) is multiplied by $K/2$ in order to determine a bound for . However, to obtain a more stringent bound, we can consider bounds of other cluster pairs, and summarize all bounds as follows:

$$differenceBound \leq \sum_{i=1}^{\frac{M}{2g}} \left(sortList_{(M-ig)} - sortList_{ig} \right) \tag{17}$$

The *sortList* indicates the load list of the controllers sorted in ascending order, M. In table III we show a summary of the time complexity of each of the algorithms we developed in this paper. Explanation on how each time complexity has been derived can be found in the corresponding sections.

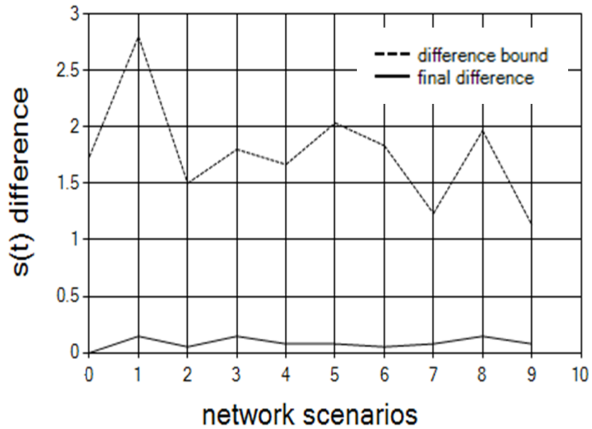

Fig. 6: Difference bound and the final difference results

VI. Simulations

We simulated a network, with several clusters and one super controller. The controllers are randomly deployed over the network. The number of flows (the controller load) of each controller is also randomly chosen. The purpose of the simulation is to show that our DCC algorithm (see section V-D) meets the difference bound (defined in section V-E), and the number of replacements bound while providing better results than the fixed clustering method. The simulator we used has been developed with Visual Studio environment in .Net . This simulator enables to choose the number of controllers, the distance between each of them and the loads on each controller. In order to consider a global topology the simulator enables also to perform a random deployment of the controllers and also to allocate random load on each controller. We used this latter option to generate each scenario in the following figures.

First we begin by showing that the bound for the $s(t)$ function is met.We used 30 controllers divided into 5 clusters. We ran 60 different scenarios. In each scenario, we used a random topology, and random controllers' loads. Figure 6 shows the optimality bound (Eq. 15), which appears as a dashed line, and the actual results for the differences achieved after all replacements. For each cluster, we chose randomly a minimalNumber in the range [20,10000], and set the controllers' loads for this cluster randomly in a range of [minimalNumber, innerBalance]. We set the innerBalance to 40. X In such a way, we get unbalance between clusters, and a balance in each cluster. The balance in each cluster simulated the master operation. The innerBalance set the quality of the balance inside the cluster. Our algorithm balanced the load between clusters and showed the different results that indicates the quality of the balance. The distances between controllers were randomly chosen in a range of [1,100].

We ran these simulations with different clusters size of: 2,3,5,10 and 15. The results showed that when the cluster size increases, the distance of the different bound from the actual bound also increases. We can also see that when the cluster size is too big (15) or too small (2) the final results are less balanced. The reason is because too small cluster size does not contain enough controllers for flexible balancing, and too big cluster do not allow flexibility between clusters since it decreases the number of clusters.

We got similar results when running 50 controllers with cluster sizes: 2,5,10,25.

As the number of controllers increases, the distance between the difference bound and the actual difference increases. This is because the bound is calculated according to the worst case scenario. Figure 7 shows the increase in distance between the actual difference distance from the difference bound when the number of controllers increases. The results are for 5 controllers in a cluster with 50 network scenarios.

We now refer to the number of replacements required. As shown in Figure 8, the actual replacement number is lower than the bound. The results are for 30 controllers and 10 clusters over 40 different network scenarios (as explained above for fig. 6).

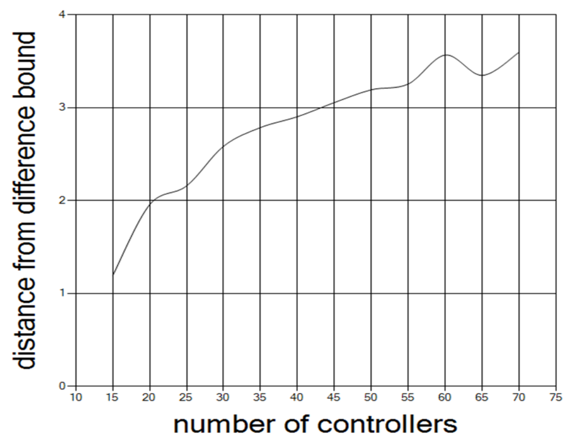

Fig. 7: Distance between the difference bound and actual difference

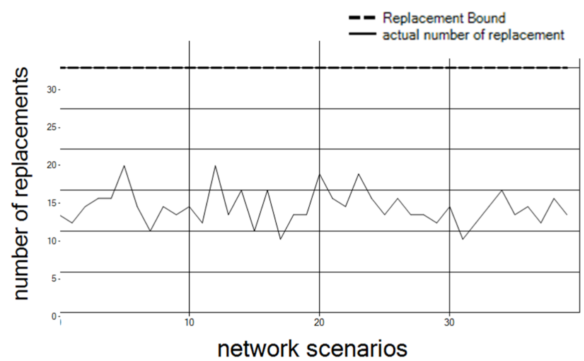

Fig. 8: Replacements bound and actual number of replacements

The number of clusters affects the number of replacements. As the number of clusters increases, the number of replacements increases. Figure 9 shows the average number of replacements over the 30 network configurations, with 100 controllers, where the number of clusters increases.

As noted, the initialization of step 1 in the DCC algorithm reduces the number of replacements required in step 2. Figure 10 depicts the number of replacements required, with and without initialization of step 1. The results are for 75 controllers and 15 clusters over 50 different network scenarios.

As mentioned previously in Section V-A1 during the initialization we can consider also the constraint on the distance (although it is not mandatory and in Section V-A2 we presented an initialization based on load only). Thus, if a controller-to-controller maximal distance constraint is important, we have to compute the lower bound on the maximal distance. By adding this lower bound to the offset defined by the user, an upper bound called "*Cnt*" is calculated (Eq.8). Figure 11 shows the final maximal distance that remains within the upper and lower bounds. The results are for 30 controllers and 10 clusters with offset 20 over 30 network scenarios.

Finally, we compare our method of dynamic clusters with another method of fixed clusters. As a starting point, the controllers are divided into clusters according to the distances between them (heuristic 2). In each time cycle, the clusters are rearranged according to the controllers' loads of the previous time cycle. The change in the load status from cycle to cycle is defined by the following transition function:

$$f(n) = \left\{ \begin{array}{ll} max(l(t)_i + random(range), P) & random(0,1) = 1 \\ max(l(t)_i + random(range), 0) & else \end{array} \right\}$$

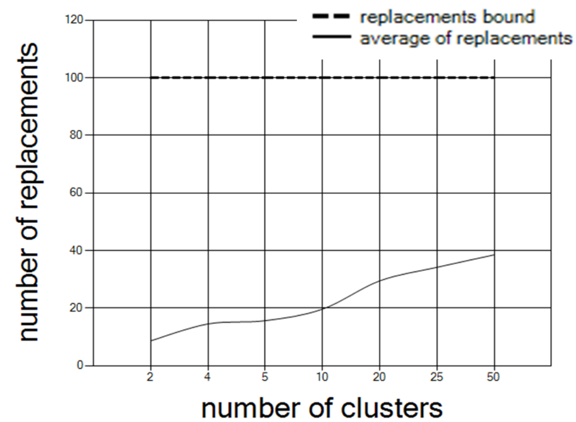

Fig. 9: Increase in the number of replacements with an increase in the number of clusters

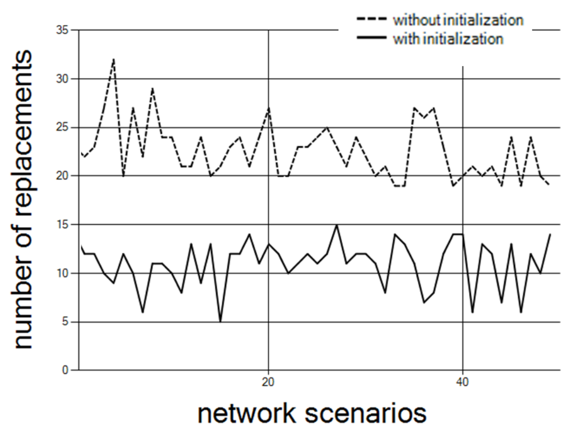

Fig. 10: Number of replacements with and without initialization

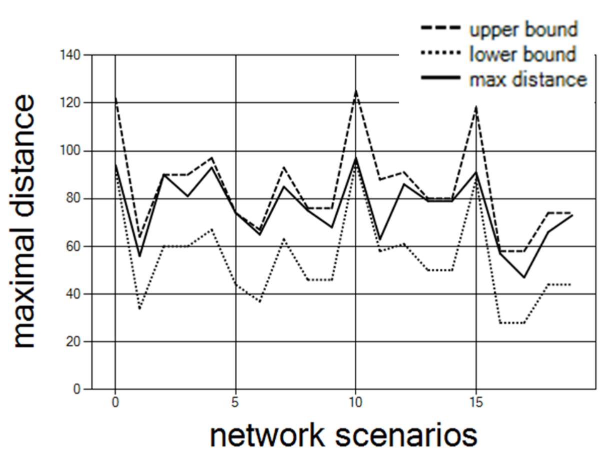

Fig. 11: Maximal distance between lower and upper bounds

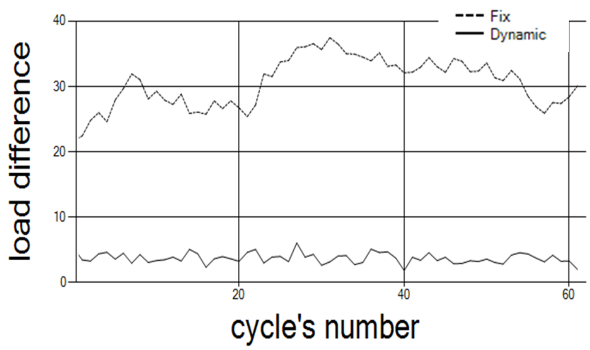

Fig. 12: Dynamic clusters vs. fix clusters comparing results

No.of controllers	No.of clusters	Cycles	Fixed clustering	Dynamic clustering	Improvement Factor
16	4	20	319.81	61.64	5.2
44	11	29	999.09	185.33	5.3
70	14	18	1501.95	267.33	5.6
42	14	28	1044.48	209.15	4.9
30	6	22	610	109	5.6

TABLE IV: Dynamic Vs Fixed Clustering in different network topologies

where P is the number of requests per second a controller can handle. The load in each controller increases or decreases randomly. We set the range at 20, and P at 1000. Figure 12 depicts the results with 50 controllers partitioned into 10 clusters. The results show that the differences between the clusters' loads are lower when the clusters are dynamic.

Following Fig 12, we ran simulation (see Table IV) for different configuration than the one in fig 12, where 50 controllers were considered. Here, we simulated the comparison on different random topologies, with a different number of controllers and clusters in each topology. The number of cycles we run is also randomly chosen. The simulation results indicate that the difference is improved fivefold by the dynamic clusters in comparison to the fixed clusters.

VII. Conclusion

In this paper, an improved approach to reduce the time complexity of the load balancing in the SDN control plane is presented. The goal is to split the requests (from the switches to controllers) among different controllers in order to avoid overload on some of them. For this purpose, we leverage a three-tier control plane architecture with a Super Controller and Master controllers, which can perform load-balancing action independently. Therefore the whole load balancing process can be executed in parallel and reduce the time complexity.

We propose a system (made of multiple algorithms) that assign controllers to clusters with an optimization and the maximal distance between two controllers in the same clusters.

We show that using dynamic clusters provide better results than fixed clustering.

In future research, we plan to explore the optimal cluster size, and allow clusters of different sizes. An interesting direction concerns overlapping clusters. Another direction is to examine the required ratio between the runtime of the load balancing algorithm and the length of the unit on the timeline. Finally, optimal placement of the master controllers in each cluster is also an important open issue.

Acknowledgement

This research was (partly) funded by the Israel Innovations Authority under the Neptune generic research project. Neptune is the Israeli consortium for network programming.

References

[1] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "Sdn security: A survey," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN For*. IEEE, 2013, pp. 1–7.

[2] T. Hu, Z. Guo, P. Yi, T. Baker, and J. Lan, "Multi-controller based software-defined networking: A survey," *IEEE Access*, vol. 6, pp. 15 980–15 996, 2018.

[3] Y. Zhang, L. Cui, W. Wang, and Y. Zhang, "A survey on software defined networking with multiple controllers," *Journal of Network and Computer Applications*, vol. 103, pp. 101 – 118, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804517303934

[4] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 7–12.

[5] J. Lu, Z. Zhang, T. Hu, P. Yi, and J. Lan, "A survey of controller placement problem in software-defined networking," *IEEE Access*, pp. 1–1, 2019.

[6] Y. Hu, W. Wendong, X. Gong, X. Que, and C. Shiduan, "Reliability-aware controller placement for software-defined networks," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 2013, pp. 672–675.

[7] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136–141, 2013.

[8] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, 2010.

[9] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann, "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.

[10] G. Saadon, Y. Haddad, and N. Simoni, "A survey of application orchestration and oss in next-generation network management," *Computer Standards & Interfaces*, vol. 62, pp. 17 – 31, 2019.

[11] S. Auroux, M. Draxler, A. Morelli, and V. Mancuso, "Dynamic network reconfiguration in wireless densenets with the crowd sdn architecture," in *Networks and Communications (EuCNC), 2015 European Conference on*, June 2015, pp. 144–148.

[12] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed sdn controller," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 7–12.

[13] A. Krishnamurthy, S. P. Chandrabose, and A. Gember-Jacobson, "Pratyaastha: an efficient elastic distributed sdn control plane," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 133–138.

[14] Y. Liu, A. Hecker, R. Guerzoni, Z. Despotovic, and S. Beker, "On optimal hierarchical sdn," in *Communications (ICC), 2015 IEEE International Conference on*. IEEE, 2015, pp. 5374–5379.

[15] Y. Fu, J. Bi, Z. Chen, K. Gao, B. Zhang, G. Chen, and J. Wu, "A hybrid hierarchical control plane for flow-based large-scale software-defined networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 2, pp. 117–131, 2015.

[16] P. D. Bhole and D. D. Puri, "Distributed hierarchical control plane of software defined networking," in *Computational Intelligence and Communication Networks (CICN), 2015 International Conference on*. IEEE, 2015, pp. 516–522.

[17] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[18] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks," in *Network and Service Management (CNSM), 2013 9th International Conference on*. IEEE, 2013, pp. 18–25.

[19] B. Görkemli, A. M. Parlakışık, S. Civanlar, A. Ulaş, and A. M. Tekalp, "Dynamic management of control plane performance in software-defined networks," in *NetSoft Conference and Workshops (NetSoft), 2016 IEEE*. IEEE, 2016, pp. 68–72.

[20] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "Balanceflow: controller load balancing for openflow networks," in *Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on*, vol. 2. IEEE, 2012, pp. 780–785.

[21] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic sdn controller assignment in data center networks: Stable matching with transfers," in *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE, 2016, pp. 1–9.

[22] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. R. Kompella, "Elasticon; an elastic distributed sdn controller," in *Architectures for Networking and Communications Systems (ANCS), 2014 ACM/IEEE Symposium on*. IEEE, 2014, pp. 17–27.

[23] H. Yao, C. Qiu, C. Zhao, and L. Shi, "A multicontroller load balancing approach in software-defined wireless networks," *International Journal of Distributed Sensor Networks*, vol. 11, no. 10, p. 454159, 2015.

[24] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[25] H. Sufiev and Y. Haddad, "Dcf: Dynamic cluster flow architecture for sdn control plane," in *Consumer Electronics (ICCE), 2017 IEEE International Conference on*. IEEE, 2017, pp. 172–173.

[26] ——, "A dynamic load balancing architecture for sdn," in *Science of Electrical Engineering (ICSEE), IEEE International Conference on the*. IEEE, 2016, pp. 1–3.

[27] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern recognition*, vol. 36, no. 2, pp. 451–461, 2003.

[28] J. P. Kahan and A. Rapoport, *Theories of coalition formation*. Psychology Press, 2014.

[29] T. M. Apostol, *Introduction to analytic number theory*. Springer Science & Business Media, 2013.

[30] D. S. Hochbaum, *Approximation algorithms for NP-hard problems*. PWS Publishing Co., 1996.

[31] A. Lim, B. Rodrigues, F. Wang, and Z. Xu, "k-center problems with minimum coverage," in *International Computing and Combinatorics Conference*. Springer, 2004, pp. 349–359.

Hadar Sufiev received her B.Sc. in Software Engineering and Teaching certificate for science and technology for high school , from the Jerusalem College of Technology in 2009. She recently received her M.Sc diploma in computer science from the Open University of Israel in Sept. 2017. She is now a lecturer at the Jerusalem College of Technology (JCT) in Jerusalem, Israel. Hadar's current research interests include optimization of the SDN control plane.

Yoram Haddad received his BSc, Engineer diploma and MSc (Radiocommunications) from SUPELEC (leading engineering school in Paris, France) in 2004 and 2005, and his PhD in computer science and networks from Telecom ParisTech in 2010. He was a Kreitman Post-Doctoral Fellow at Ben-Gurion University, Israel between in 2011-2012. He is actually a tenured senior lecturer at the Jerusalem College of Technology (JCT) in Jerusalem, Israel. Yoram's published dozens of papers in international conferences (e.g. SODA,..) and journals. He served on the Technical Program Committee of major IEEE conferences and served as a reviewer for top tier Journals. He is the recipient of the Henry and Betty Rosenfelder outstanding researcher award for year 2013. Yoram's main research interests are in the area of Wireless Networks and Algorithms for networks. He is especially interested in energy efficient wireless deployment, modeling of wireless networks, device-to-device communication, Wireless Software Defined Networks (SDN) and technologies toward 5G cellular networks.

Leonid Barenboim is a senior lecturer in the Computer Science division of the Open University if Israel. He held post-doctoral positions in the Simons Institute at UC Berkeley and the Weizmann Institute of Science. He obtained his PhD from Ben-Gurion University of the Negev. His research interests include Wireless Networks, Distributed Algorithms, Dynamic Algorithms, and Big Data. He is a (co)author of a monograph and a variety of scientific papers in these fields.

Jose Soler is Associate Professor in the Networks Technology and Service Platforms group, at DTU Fotonik, Technical University of Denmark. MSc in Telecommunication Engineering from Zaragoza University (Spain) in 1999 and PhD degree in Electrical Engineering from DTU (Denmark) in 2005. He holds also an MBA from UNED (2016) and a degree in Management from Erhversakademiet Copenhagen Business (2010). Previous employee of ITA (Spain), ETRI (South Korea), COM DTU (Denmark) and GoIP International (Denmark). His research interests include integration of heterogeneous telecommunication networks and telecommunication software and services. He serves as TPC member in an extended number of conferences and journal review panels and has participated in applied-research projects since 1999.