



Dynamic SDN Controller Load Balancing

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

Published in:
Future Internet

Link to article, DOI:
[10.3390/fi11030075](https://doi.org/10.3390/fi11030075)

Publication date:
2019

Document Version
Peer reviewed version

[Link back to DTU Orbit](#)

Citation (APA):
Sufiev, H., Haddad, Y., Barenboim, L., & Soler, J. (2019). Dynamic SDN Controller Load Balancing. *Future Internet*, 11, Article 75. <https://doi.org/10.3390/fi11030075>

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

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 multi-tier architecture for SDN control plane which can adapt itself to dynamic traffic load and therefore provides a dynamic load balancing.

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

50 II. RELATED WORKS

51 The problems of how to provide enough controllers to satisfy the traffic demand, and where to place
 52 them, were studied in [12], [13], [4]. The controllers can be organized hierarchically, where each controller
 53 has its own network sections that determines the flows it can serve [14], [15], [16], or in a flat manner
 54 where each controller can serve all types of incoming requests [17], [18], [19]. In any case every switch
 55 needs a primary controller (it can as well have more, as secondary/redundant). In most network $N \gg M$,
 56 where N is the number of switches and M is the number of controllers, Therefore, each controller has a
 57 set of switches that are linked to it. The dynamic requests rate from switches can create a bottleneck at
 58 some controllers because each controller has limited processing capabilities. Therefore if the number of
 59 switches request is too large, the requests will have to wait in the queue before being processed by the
 60 controller which will cause long response times for the switches. To prevent the aforementioned issue,
 61 switches are dynamically reassigned to controllers according to their request rates [18], [20], [21]. This
 62 achieves a balance between the loads that the controllers have.

63 In general, these load-balancing methods split the timeline into multiple time slots (TSs) in which the
 64 load balancing methods are executed. At the beginning of each TS, these methods propose to run a load
 65 balancing algorithm based on the input gathered in the previous TS. Therefore, these methods assume the
 66 input is also relevant for the current TS. The load-balancing algorithm is executed by a central element
 67 called the Super Controller (SC). Some of the methods presented in the literature are adapted to dynamic
 68 traffics [19], [18]. They suggest changing the number of controllers and their locations, turning them on and
 69 off, in each cycle according to the dynamic traffic. In addition to load balancing, some other methods [6],
 70 [18] deal with additional objectives such as minimal set-up time and maximal utilization, which indirectly
 71 help to balance loads between controllers.¹ Changing the controller location causes reassignment of all its
 72 switches, thus, such approaches are designed for networks where time complexity is not a critical issue.

73 However, in our work, we do consider time sensitive networks, and therefore, we adopted a different
 74 approach that causes less noise in the network, whereby the controllers remain fixed and the reassignment
 75 of switches is performed only if necessary in the ongoing TS (as detailed further in this paper).

76 In [18], [20], [21] the SC run the algorithm that reassigns switches according to the dynamic information
 77 (e.g. switch requests per second) it gathers each time cycle (finding the optimal time cycle duration is
 78 the goal of our future works and is not considered in this paper) from all controllers, and changes the
 79 default controllers of switches. Note that each controller should publish its load information periodically
 80 to allow SC to partition the loads properly.

81 In [20] a load balancing strategy called “Balance flow” focuses on controller load balancing such that (1)
 82 the flow-requests are dynamically distributed among controllers to achieve quick response, (2) the load
 83 on an overloaded controller is automatically transferred to appropriate low-loaded controllers to maximize
 84 controller utilization. This approach requires each switch to enable to get service from some controllers
 85 for different flow. The accuracy of the algorithm is achieved by splitting the switch load between some
 86 controllers according the source and destination of each flow.

87 DCP-GK and DCP-SA, are greedy algorithms for Dynamic Controller Placement (DCP) from [18], which
 88 employ for the reassignment phase, Greedy Knapsack (GK) and Simulated Annealing (SA) respectively,

¹The 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	$O(\max((N^2) \cdot \log(N^2), N^2 \cdot M))$	$O(M \cdot N \cdot \log N)$	$O(M \cdot N \cdot \log N)$	$O(N \cdot M^2)$

TABLE I: Time complexity comparison

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

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

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

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

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

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

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

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

125 Towards that target, the DCF architecture has been updated to enable the application of existing
126 algorithms [18], [20], [21], [23] in the "Reassignment" process in each cluster. Furthermore a clustering
127 algorithm has been developed that takes into account the controller-to-controller distances in the load
128 balancing operation. The problem has been formulated as an optimization problem, aiming at minimizing
129 the difference between cluster loads with constraints on the controller-to-controller distance. This is a
130 challenging problem due to these opposite objectives. We assume that each controller has the same limit
131 capacity in terms of requests per second that it can manage. The controllers are dynamically mapped to
132 clusters when traffic changes. The challenge is to develop an efficient algorithm for the mentioned problem
133 i.e. re-clustering in response to variations of network conditions, even in large-scale SDN networks. We

134 propose a novel **Dynamic Controllers Clustering** (further denoted as DCC) algorithm by defining our
 135 problem as a K-Center problem at the first phase and developing, in a second phase, a replacement
 136 rule to swap controllers between clusters. The idea for the second phase is inspired by Game Theory.
 137 The replacements shrink the gap between cluster loads while not exceeding the constraint of controller-
 138 to-controller distances within the cluster. We assume that M controllers are sufficient for handling the
 139 maximal request rate in the network (as mentioned earlier, there are already many works that found the
 140 optimal number of controllers).

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

143 III. NETWORK ARCHITECTURE

144 A. *Dynamic Cluster Flow Architecture*

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

156 B. *Three level load balancing architecture*

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

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

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

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

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

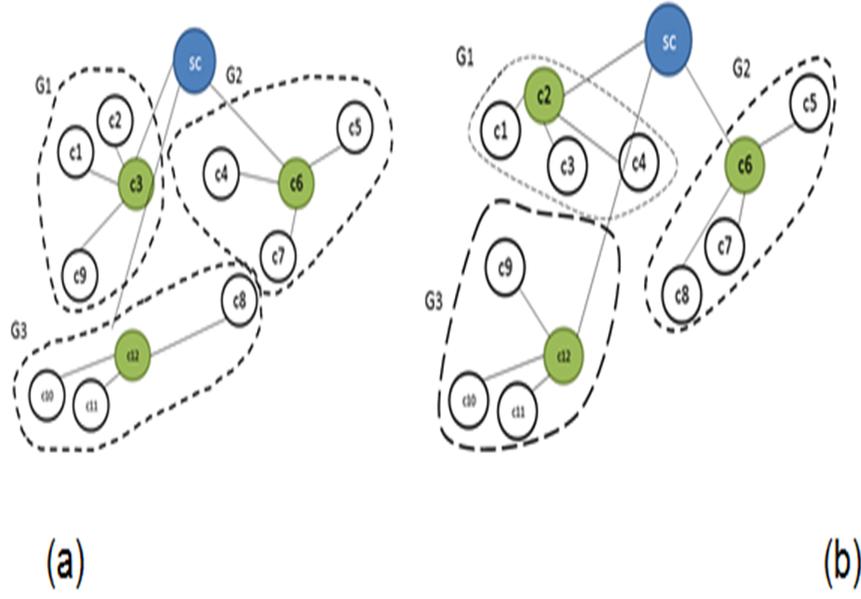


Fig. 1: Three-tier control plane

IV. DCC PROBLEM FORMULATION

181

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

A. Notations

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

193 Figure 2 shows an example of such a matrix $Y(9 \times 3)$ corresponding to nine controllers split into three
 194 clusters. On the left part we see Y matrix before re-clustering process and on the right after it is completed.
 195 Thanks to Y one can know which controller is in which cluster as follows. If a controller is included in a
 196 cluster then in the corresponding row of the controller and corresponding column of the cluster the value
 197 will be 1 otherwise it will be 0. For instance, we see that before re-clustering controller number 5 is in
 198 cluster b. After re-clustering (on the right matrix) we see that controller 5 is no longer in b but has been
 199 moved to cluster c.

200 Therefore, Y is a binary $M \times K$ matrix as follows:

$$Y(t)_{ij} = \begin{cases} 0 & C_j \in G_i \\ 1 & \text{else} \end{cases} \quad (1)$$

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

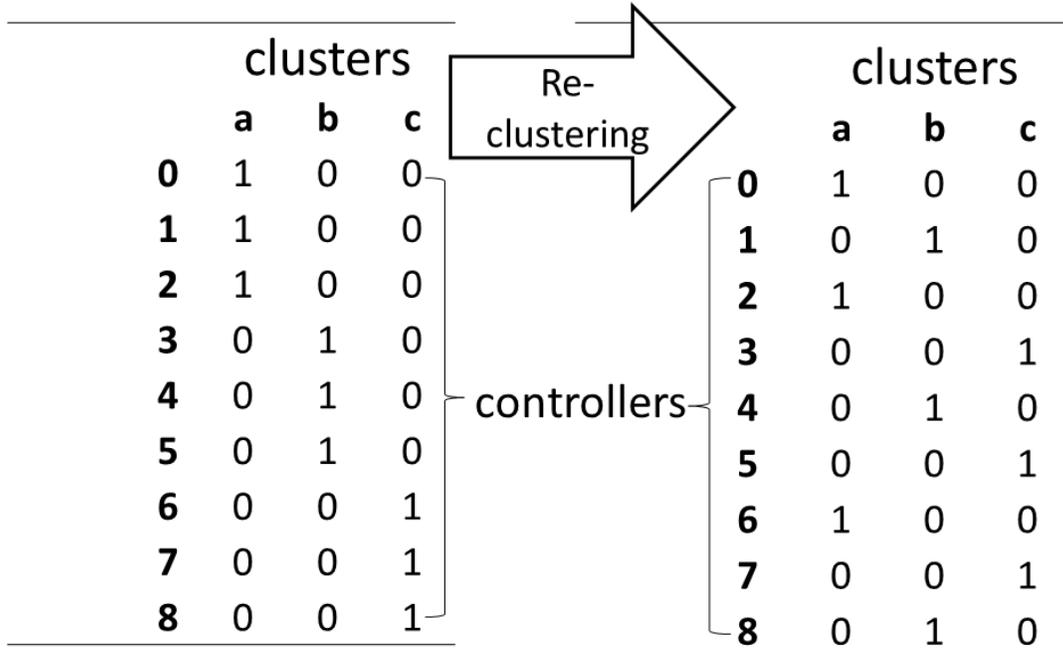


Fig. 2: Y matrix examples before and after re-clustering

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 $Y(t)_{ji} = 0$
$l(t)_j$	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

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

206 B. Clusters' Load Differences

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

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

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

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

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

$$\vartheta(t)_i = |\theta(t)_i - Avg| \quad (5)$$

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

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

218 C. Distances between controllers within same cluster

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

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

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

$$Cnt = \text{maxDistance} + \text{offset} \quad (8)$$

234 D. Optimization Problem: Dynamic Controllers' Clustering

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

$$\text{Minimize } \zeta(t) \quad (9)$$

238 subject to:

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

239

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

240

$$\eta(t) < Cnt \quad (12)$$

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

241 Equation 10 ensures that each cluster has exactly M/K controllers at a given time while Equation 11
 242 ensures that each controller is assigned to exactly one cluster at a time. Equation 12 puts a constraint on
 243 the maximum distance between controllers within same cluster.

244 Regarding the distance constraint, the problem is a variant of a k -Center problem [27].

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

247 These two general problems are NP-Complete because finding an optimal partition requires iterating
 248 over all the partitions of the player set, where the number of these partitions grows exponentially with the
 249 number of players, and is given by a value known as the Bell Number [29]. Hence, finding an optimal
 250 partition in general is computationally intractable and impractical (unless $P = NP$).

251 In this paper, we propose an approximation algorithm to solve these problems. We adapt the K -Center
 252 problem solution for initial clustering, and use game theoretic techniques to satisfy our objective function
 253 with the distance constraint.

254

V. DCC TWO PHASE ALGORITHM

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

261 A. Phase 1: Initial Clustering

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

266 1) *Initial clustering with the distance constraint:* Most of the control messages concerning the cluster
 267 load balancing operation are generated due to the communication between the controllers and their MC.
 268 Thus, we use the K -Center problem solution to find the closer MC [30], [27]. In this problem, $C =$
 269 $\{C_1, C_2, \dots, C_K\}$ is the center's set and $P = \{P_1, P_2, \dots, P_M\}$ contains M controllers. We define $P_C =$
 270 $(d(p_1, C), d(p_2, C), \dots, d(p_M, C))$, where the i^{th} coordinate of P_C is the distance of p_i to its closest
 271 center in C . The k -Center inputs are: a set P of M points and an integer number K , where $M \in N$,
 272 $K < M$. The goal is to find a set of k points $C \subseteq P$ such that the maximum distance between a point in P
 273 and its closest point in C is minimized. The network is a complete graph, and the distance definition [see
 274 Table II] satisfies the triangle inequality. Thus, we can use an approximate solution to the k -Center problem
 275 to find MCs. Given a set of centers, C , the k -center clustering price of P by C is $\|P_C\|_\infty = \max_{p \in P} d(p, C)$.
 276 Algorithm 1 is an algorithm similar to the one used in [31]. This algorithm computes a set of k centers,

277 with a 2-approximation to the optimal k -center clustering of P , i.e., $\|P_K\|_\infty \leq 2opt_\infty(P, K)$ with $O(MK)$
 278 time and $O(M)$ space complexity [31].

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

input: $P = \{p_1, p_2, \dots, p_M\}$ controllers set, controller-to-controller matrix distances

output: $C = \{c_1, c_2, \dots, c_K\}$ 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:      $d_i[p] \leftarrow \min_{c \in C} d(p, c)$ 
7:   end for
8:    $c_i \leftarrow \max_{p \in P} d_i[p]$ 
9:    $C \leftarrow C \cup c_i$ 
10: end for
11: return  $C$  // The master set

```

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

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

286 As depicted in Heuristic 2, lines 1-2 prepare set S that contains the list of controllers to assign. lines 3-5
 287 define the initial empty clusters with one master for each one. Lines 7-15 (while loop) are the candidate
 288 clusters which have less than M/K controllers, and each controller is assigned to the nearest master of
 289 these candidates. After the controllers are organized into clusters, we check the maximal distance between
 290 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: $C = \{c_1, c_2, \dots, c_M\}$ Controller list $M = \{m_1, m_2, \dots, m_k\}$ masters list

controller-to-controller matrix distances

output: $CL = \{cl_1, cl_2, \dots, cl_k\}$ Clusters list, where $CL_i = \{cl_{i1}, cl_{i2}, \dots, cl_{i(m/k)}\}$ $maxDistance$ - maximum distance between controllers in a cluster.**procedure:**

```

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

```

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

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

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

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

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

310 where

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

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

315

Heuristic 3 Load initialization

input:

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

Masters $\bar{C}VL_i$'s (average flow-request number (loads) for each controller)

integer K for number of clusters

output:

$P = \{p_1, p_2, \dots, p_K\}$ Clusters list, where $P_i = \{c_{i1}, c_{i2}, \dots, c_{i(M/K)}\}$

procedure:

- 1: $SortedListC \leftarrow$ descending order of controllers list according to their loads
 - 2: $Candidates \leftarrow P$
 - 3: **for all** $c \in SortedListC$ **do**
 - 4: $P_{min} \leftarrow$ the cluster with minimal $Cost_g(C)$ from candidates
 - 5: $P_{min} \leftarrow P_{min} \cup c$
 - 6: **if** $|P_{min}| = M/K$ **then**
 - 7: $Candidates \leftarrow Candidates - P_{min}$
 - 8: **end if**
 - 9: **end for**
 - 10: **return** P
-

316 B. Initial Clustering as Input to the Second Phase

317 The outcomes of the two types of initialization, namely "distance" and "load", presented so far (section
 318 V-A) are used as an input for the second phase.

319 It should be noted that since the "maxDistance" constraint is an output of the initialization based on the
 320 distance (Heuristic 2), the first phase is mandatory in case the distance constraint is tight. On the other
 321 hand, the initialization based on the load (Heuristic 3) is not essential to being perform load balancing in
 322 the second phase, but it can accelerate convergence in the second phase.

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

324 In the second phase, we apply the coalition game theory [28]. We can define a rule to transfer participants
 325 from one coalition to another. The outcome of the initial clustering process is a partition denoted Θ
 326 defined on a set C that divides C into K clusters with M/K controllers for each cluster. Each controller is
 327 associated with one cluster. Hence, the controllers that are connected to the same cluster can be considered
 328 participants in a given coalition.

329 We now leverage the coalition game-theory in order to minimize the load differences between clusters or
 330 to improve it if an initial load balancing clustering has been performed such as in V-A2

331 A coalition structure is defined by a sequence $B = \{B_1, B_2, \dots, B_l\}$ where each B_i is a coalition. In
 332 general, a coalition game is defined by the triplet (N, v, B) , where v is a characteristic function, N are the
 333 elements to be grouped and B is a coalition structure that partitions the N elements [28]. In our problem
 334 the M controllers are the elements, G is the coalition structure, where each group of controllers G_i is
 335 a coalition. Therefore, in our problem we can define the coalition game by the triplet (M, v, G) where
 336 $v = \zeta(t)$. The second phase can be considered as a coalition formation game. In a coalition formation
 337 game each element can change its coalition providing this can increase its benefit as we will define in
 338 the following. For this purpose, we define the Replacement Value (RV) as follows:

339

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

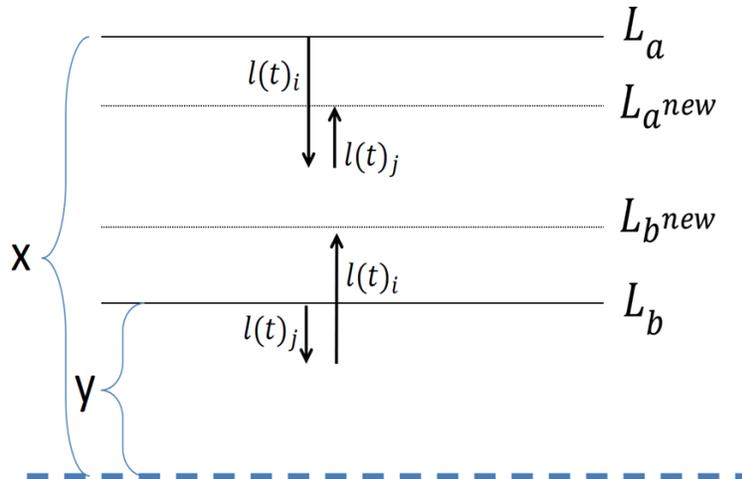


Fig. 3: Clusters loads after replacement on the same side with reference to the average

340 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 $(\vartheta(t)_{a^{old}} \leq$
 341 $Avg) \& (\vartheta(t)_{b^{old}} \leq Avg)$ and *aboveAverage* is true where $(\vartheta(t)_{a^{old}} \geq Avg) \& (\vartheta(t)_{b^{old}} \geq Avg)$. Each replacement
 342 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
 343 loads L_a and L_b , respectively. We use the notations "old" and "new" to indicate a value before and after
 344 the replacement.

345

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

349

350 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),
 351 $\varsigma(t)_{old} = \varsigma(t)_{new}$ (see Equation 6). When one of the cluster's load moves to another side of the global
 352 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
 353 set to zero.

354

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

357 In Figure 3, the sum of the loads' distances from the global average, before the replacement is $x+y$. After
 358 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,
 359 the result is the same.

360 In Figure 4 the sum of distances from the global average, before the replacement is $x + y$, and this sum
 361 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
 362 result is the same.

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

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

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

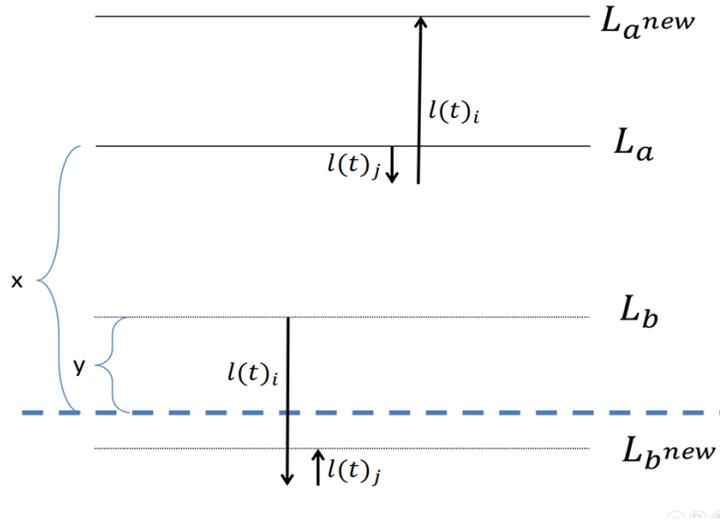


Fig. 4: Clusters' loads after replacement on different sides with reference to the average

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

373

374 Regarding the time complexity of lines 1 in algorithm 4, i.e., find the best replacement, it takes:

$$375 \frac{M}{k}(k-1) + \frac{M}{k}(k-2) + \dots + \frac{M}{k}(k-(k-1)) = \frac{M^2}{k^2} \cdot \frac{k(k-1)}{2} = \frac{M^2(k-1)}{2k} = o(M^2) \text{ time.}$$

376

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

380

Procedure 4 Replacements

input:

$CL = \{cl_1, cl_2, \dots, cl_K\}$ Clusters list, where $CL_i = \{cl_{i1}, cl_{i2}, \dots, cl_{i(M/K)}\}$

distance constraint Cnt

output:

$CL = \{cl_1, cl_2, \dots, cl_K\}$ 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
-

381 D. Dynamic Controller Clustering Full Algorithm

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

Algorithm 5 DCC Algorithm

input:

nt Network contain $C = \{c_1, c_2, \dots, c_M\}$ Controller list, and distances between controllers.
 K and M for the number of clusters and controllers, respectively
 $constraintActive$ to indicate that it meets the controller-to-controller distance constraint
 $offset$ to calculate the distance constraint (optional).

output:

$P = \{p_1, p_2, \dots, p_k\}$ Clusters list, where $P_i = \{c_{i1}, c_{i2}, \dots, c_{i(m/k)}\}$

procedure:

```

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

```

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

389
390 The DCC can run the second option without any distance constraint (Line 6). In Line 11 it chooses the
391 best solution in such cases, (referring to the minimal load differences) from the following three options:
392 (1) Partition by loads only (Line 8); (2) Start partition by loads and improve with replacements (Line 9)
393 (3) Partition by replacements only (using the previous cycle partition) (Line 10).

394
395 Regarding the time complexity, DCC uses heuristic 2, heuristic 3, algorithm 1 and algorithm 4, thus it
396 has a $O(M^3)$ time complexity.

397 E. Optimality Analysis

398 In this section, our aim is to prove how close our algorithm is to the optimum. Because the capacity of
399 controllers is identical, the minimal difference between clusters is achieved when the controllers' loads are
400 equally distributed among the clusters, where the clusters' loads are equal to the global average, namely
401 $\zeta(t) = 0$. Since in the second phase, i.e., in the replacements, the DCC full algorithm is the one that sets
402 the final partition and therefore determines the optimality, it is enough to provide proof of this.
403 As mentioned before, the replacement process is finished when all RV 's are 0, at which time any
404 replacement of any two controllers will not improve the result. Figure 5 shows the situation for each
405 two clusters at the end of the algorithm.

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

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

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

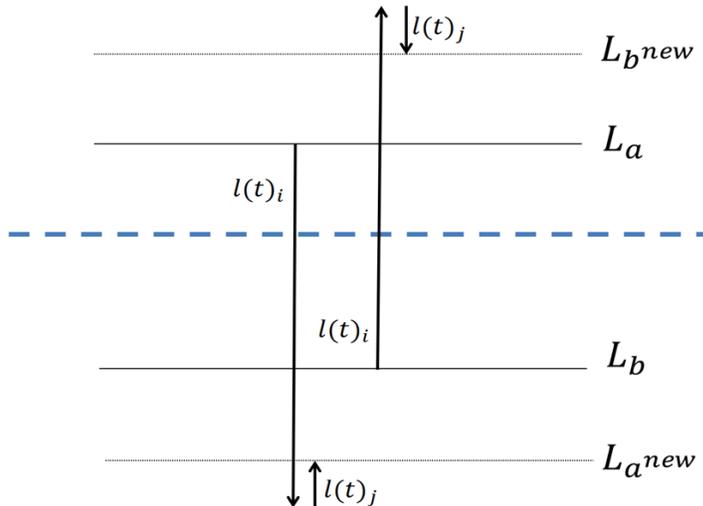


Fig. 5: The loads of each two clusters at the end of all replacements

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

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

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

$$\vartheta(t)_{most_loaded} + \vartheta(t)_{most_under_loaded} \leq l(t)_{g_th_bigger} - l(t)_{g_th_smaller} \quad (16)$$

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

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

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

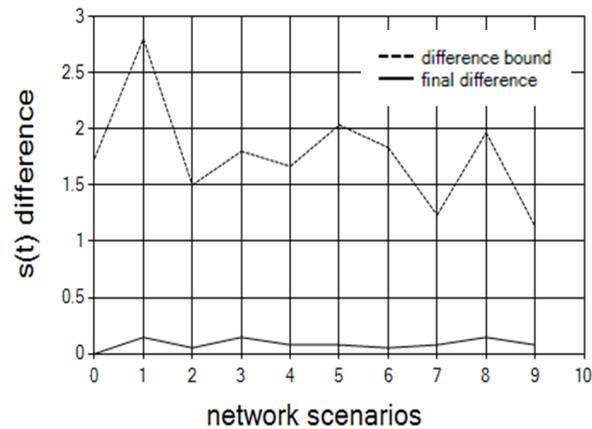


Fig. 6: Difference bound and the final difference results

VI. SIMULATIONS

428

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

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

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

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

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

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

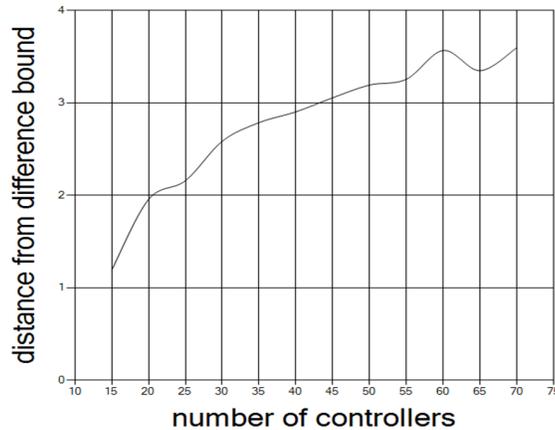


Fig. 7: Distance between the difference bound and actual difference

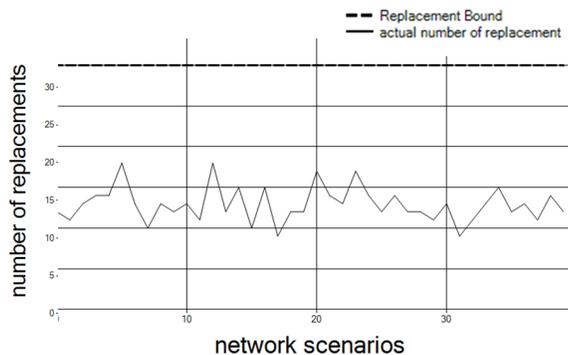


Fig. 8: Replacements bound and actual number of replacements

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

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

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

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

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

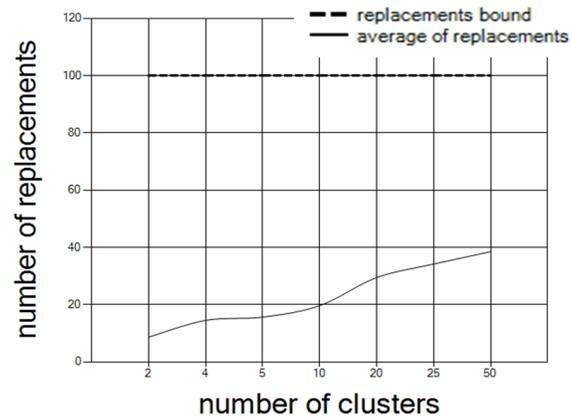


Fig. 9: Increase in the number of replacements with an increase in the number of clusters

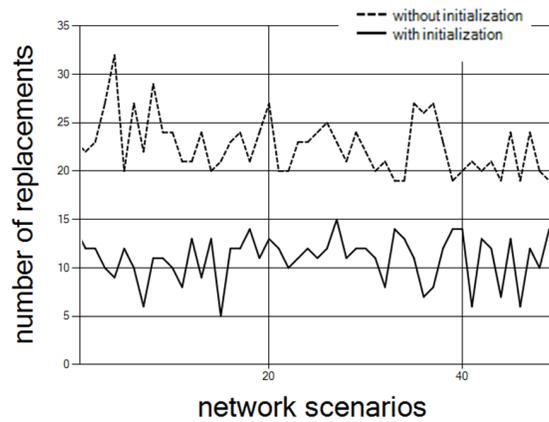


Fig. 10: Number of replacements with and without initialization

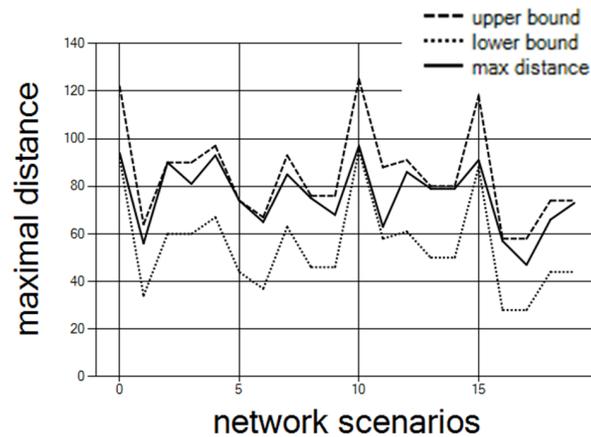


Fig. 11: Maximal distance between lower and upper bounds

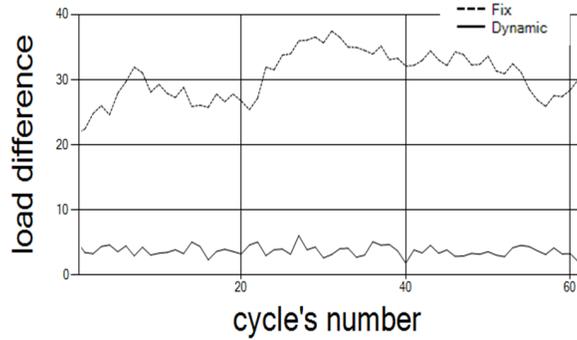


Fig. 12: Dynamic clusters vs. fix clusters comparing results

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 (CICIN), 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. Gökemli, 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.

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

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

582 **Jose Soler** is Associate Professor in the Networks Technology and Service Platforms group, at DTU Fotonik, Technical University of
583 Denmark. MSc in Telecommunication Engineering from Zaragoza University (Spain) in 1999 and PhD degree in Electrical Engineering from
584 DTU (Denmark) in 2005. He holds also an MBA from UNED (2016) and a degree in Management from Erhvervsakademiet Copenhagen
585 Business (2010). Previous employee of ITA (Spain), ETRI (South Korea), COM DTU (Denmark) and GoIP International (Denmark). His
586 research interests include integration of heterogeneous telecommunication networks and telecommunication software and services. He serves
587 as TPC member in an extended number of conferences and journal review panels and has participated in applied-research projects since
588 1999.