Distributed Resilient Control Plane for Large Software Defined Networks

Eugen Borcoci, Stefan Ghita
University POLITEHNICA of Bucharest - UPB
Bucharest, Romania
Emails: eugen.borcoci@elcom.pub.ro, stalghita@gmail.com

Abstract —**Large Software Defined Networks (SDN) solve the control scalability problem coming from the SDN control centralization principle, by defining and installing several regional controllers. Therefore, a controller placement problem (CPP) should be solved. During run-time, possible failures of links or node appear; then a forwarder node could try to select an available and reachable controller among those which are functional. This is called controller selection problem (CSP). Although many studies have been published, the above problems are still open research issues, given the various network contexts, providers' policies and possible multiple, different optimization criteria. Therefore, multi-criteria decision algorithms can provide valuable solutions. This paper is based on a previous work which has developed a simulation model for multi-controller SDN network, targeting optimization while including resilience aspects of the controller placement problem and controller selection problem. The current paper extends that work, by including a more in-depth analysis, giving relevant examples and introducing additional novel experiment results.**

Keywords — Software Defined Networking; Multi-criteria optimization; Controller placement; Controller selection; Forwarder nodes assignment; Reliability; Resilience

I. INTRODUCTION

This paper is an extended version of the work [1], which has been dedicated to study methods to optimize the *Software Defined Networking* (SDN) controller placement and selection in large area SDN networks. It is known that SDN has as basic principles the decoupling of the architectural *Control Plane* (CPl) w.r.t *Data Plane* (DPl) and also CPl centralization in SDN controllers. Therefore, in large network environments and considering the limited processing of controllers, scalability problems of the CPl appear [2]. The usual solution for this, adopted in many studies, is a distributed multi-controller implementation of the SDN control plane. Different flat or hierarchical organizations for a multi-controller SDN control plane have been developed, e.g., in [3][4].

In a basic approach, the *SDN controller* (SDN-C) is understood as a software control entity installed/placed in a geographically distinct location, i.e., a particular physical network node. The control plane is defined as an overlay network on top of the physical one. The links between controllers can be physical or virtual.

Recently, the *Network Function Virtualization* technologies [5] allow that several logical SDN-Cs could be realized as virtual entities running on top of virtual machines (notation for such controllers could be vSDN-C), i.e., several logical controllers can be collocated in the same physical node.

In this work we suppose the basic approach of the SDN controllers' implementation. However, the models developed in here can be as well applied to a virtualized environment. In such a case, the essential modification of the model (considering the optimization objective of this study) is that the logical controllers will be virtually linked through a control plane graph.

The *controller placement problem* (CPP) is a complex one, given the variety of factors involved. Some examples are: how the network topology is specified - flat or hierarchical/clustered; what criteria are considered to solve the CPP; number of controllers - predefined or not; failure-free or failure-aware metrics (e.g., considering backup controllers and node/link failures); how the DPl forwarders nodes are assigned/mapped to controllers (in static or dynamic way, i.e., depending on actual network conditions and network provider policies), and others. The evaluation of the degree of optimality of different approach can be studied on some simplified topologies – in order to compare the efficiency of approaches or, on real specific network topologies. Many studies, e.g., of Heller [6] et al. - as early study - and then others [7-9][11-19] considered various aspects and solutions of the CPP.

In a real network environment, it has been apparent that there is no unique best and universal placement rule for any SDN-controlled network. Dynamic nodes addition and deletion can happen and, in such cases, a forwarder could dynamically select an appropriate controller, if it has enough pertinent and updated information. The same situation appears when the traffic in the data plane is varying and re-assignment of forwarder to controllers is required, to avoid controller overload. This is *called controller selection problem* (CSP) and can be considered as an extension of the CPP [7].

The CPP was recognized as a non-polynomial (NP) - hard problem, mentioned in the early work of Heller et al. [6]. For such problems different approximation algorithms [8] and more pragmatic solutions have been proposed, adapted in different contexts. In particular, in SDN-controlled networks case, many optimization criteria have specific, target performance, both in the data plane and control plane, in failure-free or failure-aware approaches.

Examples of specific, individual criteria could be: to maximize the controller-forwarder or inter-controller communication throughput; reduce the latency of the path connecting them; limit the controller load imbalance; find an optimum controllers' placement and forwarder-to-controller allocation, offering a fast recovery after failures (controllers, links, nodes). Also, other specific optimization goals could be added to the above list, depending on specific context (wire-line, wireless/cellular, cloud computing and data center networks) and on some specific business targets of the Service Provider.

However, a major issue is that different optimization criteria could lead to significant different placement solutions; so, a multi-criteria global optimization could be a better trade-off approach.

The paper [9] provided a contribution on multi-criteria optimization algorithms for the CPP, not by developing specific single-criterion algorithms (many other studies already did that) but to achieve an overall optimization by applying *multi-criteria decision algorithms* (MCDA) [10]. The input of MCDA is the set of candidates (an instance of controller placement is called a candidate solution). Examples have been analyzed, on some real network topologies, proving the usefulness of the approach.

The more recent paper [1] extended the model of [9]; several reliability aware criteria have been added to the CPP solution. Also the novel CSP extension is introduced, being appropriate for a dynamic network context. It has been shown that the same basic MCDA can be applied in both static and dynamic context, but with different sets of criteria. Simulation experiments and novel results have been presented.

Note some limitations: neither work [9], nor [1] touch the problem of control plane overhead and signaling issues between the controllers when a re-configuration of the SDN network is performed. This could be the topic for additional studies. Also note that in this paper, by "large Software Defined Networks", it is actually understood networks having several SDN controllers.

The structure of this paper (extension of [1]) is described here. Section II is a short overview of related work. Section III revisits several metrics and optimization algorithms and presents some of their limitations. Section IV revisits the framework for MCDA-RL (the variant which is called "*reference level*") as a simple but powerful tool applicable to solve the CPP and CSP problems. Section V presents the implementation performed to validate the MCDA proposed model in a resiliency-oriented optimization approach, and outlines the simulation experiments performed. Section VI offers few examples of simulation results to illustrate the validity of the approach. Section VII presents conclusions and future work.

II. RELATED WORK

This short section is included mainly for guiding the reader to references. More comprehensive overviews on published work on CPP in SDN-controlled WANs are given in [11-14]. The main goal is to find those controller placements that provide high performance (e.g., low delay for controller-forwarder communications) and also create robustness to controllers and/or network failures.

An early work of Heller et al. [6] has shown that it is possible to find optimal controller placement solutions for realistic network instances, in failure-free scenarios, by analyzing the entire solution space, with off-line computations (the metric is latency). The studies [15-21] have been more focused, i.e., additionally considered the resilience as being important with respect to events like: *controller* failures, *network links/paths/nodes* failures, *controller overload* (load imbalance). The *Inter-Controller Latency* is also important and, generally, it cannot be minimized while simultaneously minimizing controller-forwarders latency; a tradeoff solution could be the answer.

The works [15][17] developed several algorithms for real topologies, aiming to find solutions for reliable. SDN control, but still keep acceptable latencies. The controller instances are chosen as to minimize connectivity losses; connections are defined according to the shortest path between controllers and forwarding devices. Muller et al. [18] eliminate some restrictions of previous studies, like: single paths, processing (in controllers) of the forwarders requests only *on-demand* and some constraints imposed on failover mechanisms. Hock et al. [16] adopted a multi-criteria approach for some combinations of the metrics (e.g., max. latency and controller load imbalance for failure-free and respectively failure use cases).

In a recent work [7], K. Sood and Y. Siang propose to extend the CPP problem into CSP, i.e., to consider the dynamics of the network and make controller selection. They explore the relationship between traffic intensity, resources requirement, and QoS requirements. It is claimed that to optimize the control layer performance, the solutions must be topology-independent and adaptive to the needs of the underlying network behaviour. They propose a topology independent framework to optimize the control layer, aiming to calculate the optimal number of controllers to reduce the workload, and investigate the placement/location of the controllers. However, their first declared objective has been not to determine the optimal placement of controllers in the network, but to motivate the CSP.

In recent papers [20][21] Y. Xu, M. Cello et al., developed dynamic forwarder/switch migration scenarios and algorithms, starting from a given switch-controller assignment and partition (based on some criteria) of the network in domains, each one controlled by a single controller. Also, a realistic assumption is considered, i.e., limited processing capacity of the controllers. During run time, if some controllers are overloaded (such events are dynamically observed by a monitoring system), then a heuristic algorithm is applied, to optimally move (re-assign) a part of the switches coordinated by that controller to other controller less loaded. In order to reduce the signaling (related to migration) between controllers, the migration is cluster-based, i.e., not a single switch is migrated but a cluster of switches are moved from an overloaded

controller, e.g., CT_i, to another less loaded controller CT_j. Thus, the algorithm realizes a controller load balancing (the name *BalCon* is coined for the algorithm [20]).

Many of the mentioned studies considered a single criterion in the optimization algorithms. In [9][1] a multi-criteria algorithm is used (applicable for an arbitrary number of decision criteria) to solve the CPP; validation of results have been presented for some real network topologies [22][23].

A recent work [24] studies the load balancing via switch migration in a network having several SDN controllers. The goal is to migrate switches from overloaded to under-loaded controllers, depending on the traffic variation. The work presents a heuristic approach to solve the switch migration problem. The advantage of the proposed solution versus other approaches is that the algorithm does not halt the search whenever a switch migration is not possible. Instead, it searches for more complex moves like swapping two switches to further improve the results.

The work in this paper is an extension of [1], with focus on optimal initial placement of the SDN controllers, considering among multi-criteria some reliability – related ones.

III. EXAMPLES OF CONTROLLER PLACEMENT METRICS AND ASSOCIATED ALGORITHMS

This section is a short presentation of a few typical metrics and optimization algorithms for CPP and CSP. A more detailed presentation of them can be found in [13]. Considering a particular metric (criterion) an optimization algorithm can be run for a given metric, as in [6][15-18].

As already stated, this paper goal is not to develop a new particular algorithm based on a given single metric, but to search for a global optimization. The individual metrics presented in this section can be embedded in a multi-criteria optimization algorithm.

The SDN-controlled network is abstracted by an undirected graph G(V, E), with V - set of nodes, E – set of edges and n=|V| the total number of nodes. The edges weights represent an additive metric (e.g., *propagation latency* [6]).

A basic metric is *d(v, c): shortest path* distance from a forwarder node $v \in V$ to a controller $c \in V$. We denote by C_i a particular placement of controllers; $C_i \subseteq V$ and $|C_i| < |V|$. The number of controllers is limited to $|C_i| = k$ for any particular placement C_i. The set of all possible placements is denoted by $C = \{C_1, C_2 \ldots\}$. Some metrics are basic, i.e., failure-free; others take into account failure events of links or nodes.

An important metric for SDN control is the *latency* between nodes. Note that, while it has a dynamic nature, in some simplified assumptions it is estimated as a static value.

A. Failure-free scenarios

- *Forwarder-to-controller latency*

In Heller's work [6], two (failure-free) metrics are defined for a given placement C_i: *Worst_case_latency* and *Average_latency* between a forwarder and a controller. In [5], the above two kinds of latencies are defined, for a particular placement C_i of controllers, where $C_i \subseteq V$ and $|C_i| \leq |V|$. The number of controllers is k for any particular placement C_i. The set of all possible placements is $C = \{C_1, C_2, \ldots.\}$. One can define, for a given placement C_i :

Average_latency:

$$L_{avg}(C_i) = \frac{1}{n} \sum_{v \in V} \min_{c \in Ci} d(v, c) \quad (1)$$

Worst_case_latency :

$$L_{wc} = \max_{v \in V} \min_{c \in C_i} d(v, c) \quad (2)$$

The optimization algorithm should find a particular placement C_{opt}, where *either average latency or the worst case latency is minimum.*

The work [8] proposes an algorithm to *maximize the number of nodes within a latency bound*, i.e., to find a placement of *k* controllers, such that they cover a maximum number of forwarder nodes, but with an upper latency bound of each forwarder latency to its controller.

- *Inter-controller latency*

The SDN controllers should inter-communicate and therefore, the inter-controller latency is important. For a given placement C_i, one can minimize the *maximum latency between two controllers*. Note that this can increase the forwarder-controller distance (latency). Therefore, a trade-off is necessary, thus justifying the necessity to apply some multi-criteria optimization algorithms, e.g., like Pareto frontier - based ones [16].

B. Failure-aware scenarios

In such scenarios controller and/or network failures events are considered. The optimization process aims now to find trade-offs to preserve a convenient behavior of the overall system in failure cases (controllers, or nodes, or links).

- *Multiple-path connectivity metrics*

If multiple paths are available between a forwarder node and a controller [9], this can exploited in order to reduce the occurrence of controller-less events, in cases of failures of nodes/links. The goal in this case is to maximize connectivity between forwarding nodes and controller instances. A special metric can be defined as:

$$M(C_i) = \frac{1}{|V|} \sum_{c \in C_i} \sum_{v \in V} ndp(v, c) \quad (3)$$

The *ndp(v,c)* is the *number of disjoint paths* between a node v and a controller c, for an instance placement C_i. An optimization algorithm should *find the placement C_{opt} which maximizes $M(C_i)$.*

- *Controller failures*

To minimize the impact of such failures, the latency-based metric should consider both the distance to the (primary) controller and the distance to other (backup) controllers. For a total number of *k* controllers, the failures can be modeled [16], by constructing a set C of scenarios, including all possible combinations of faulty controller number, from 0 of up to k - 1. The *Worst_case_latency_cf* will be:

$$L_{wc-cf} = \max_{v \in V} \max_{C_i \in C} \min_{c \in C_i} d(v,c) \quad (4)$$

The optimization algorithm should find a placement which *minimizes the expression* (4).

Note that in failure-free case, the optimization algorithm tends to rather equally spread the controllers in the network, among the forwarders. To minimize (4), the controllers tend to be placed in the center of the network, such that in a worst case, a single controller can take over all control. However, the scenario supposed by the expression (4) is very pessimistic; a large network could be split in some regions/areas, each served by a primary controller; then some lists of possible backup controllers can be constructed for each area, as in [18]. The conclusion is that an optimization trade-off should be found, for the failure-free or failure cases. A multi-criteria approach can provide the solution.

- *Nodes/links failures*

For such cases, the objective could be to find a controller placement that minimizes the number of nodes possible to enter into controller-less situations, in various scenarios of link/node failures. A realistic assumption is to limit the number of simultaneous failures at only a few (e.g., two [16]). If more than two arbitrary link/node failures happen simultaneously, then the topology can be totally disconnected and optimization of controller placement would be no longer useful.

For a *placement* C_i of the controllers, an additive integer value metric $Nlf(C_i)$ could be defined, as below: consider a failure scenario denoted by f_k, with $f_k \in F$, where F is the set of all network failure scenarios (suppose that in an instance scenario, at most two link/nodes are down); initialize $Nlf_k(C_i) = 0$; then for each node $v \in V$, add one to $Nlf_k(C_i)$ if the node v has no path to any controller $c \in C_i$ and add zero otherwise; compute the maximum value (i.e., consider the worst failure scenario). In equivalent words, the algorithm counts the nodes that have no more connectivity to any controller.

$$Nlf(C_i) = \max Nlf_k(C_i) \quad (5)$$

The optimization algorithm should find a placement *to minimize* (5), where *k should* cover all scenarios of F. It is expected that increasing the number of controllers, will decrease the *Nlf* value. However, the optimum solution based on the metric (5) could be very different from those provided by the algorithms using the latency-based metrics.

- *Load balancing for controllers*

It is desired a good balance of the node-to-controller distribution. A metric $Ib(C_i)$ will measure the degree of imbalance of a given placement C_i as the *difference between the maximum and minimum number of forwarders nodes assigned to a controller.* If the failure scenarios set S is considered, then the worst case should evaluate the maximum imbalance as:

$$Ib(C_i) = \max_{s \in S} \{ \max_{c \in C_i} n_c^s - \min_{c \in C_i} n_c^s \} \quad (6)$$

where n_c^s is the number of forwarder nodes assigned to a controller c. Equation (4) takes into account that in case of failures, the forwarders can be reassigned to other controllers and therefore, the load of those controllers will increase. An optimization algorithm should find that *placement which minimizes the expression (4).*

IV. Multi-criteria Optimization Algorithms

SDN controllers' placement and/or selection may involve several particular metrics (as summarized in Section III). If optimization algorithms for particular metrics are applied, then one can obtain different non-convergent solutions. Actually the CPP and CSP problems have naturally multi-criteria characteristics; therefore, MCDA is a good way to achieve a convenient trade-off solution.

This paper uses the same variant of MCDA implementation as in [9], i.e., the *reference level (RL) decision algorithm* [10] as a general way to optimize the controller placement, and controller selection, for an arbitrary number metrics. The MCDA-RL selects the optimal solution based on normalized values of different criteria (metrics).

The MCDA considers *m* objectives functions (whose values, assumed to be positive should be minimized). A solution of the problem is represented as a point in a space R^m of objectives; the decision parameters/variables are: v_i, i = 1, ..m, with $\forall i$, $v_i \geq 0$; so, the image of a candidate solution is $Sl_s=(v_{s1},v_{s2}, ..,v_{sm})$, represented as a point in R^m. The number of candidate solutions is S. Note that the value ranges of decision variables may be bounded by given constrains. The optimization process consists in selecting a solution satisfying a given objective function and conforming a particular metric.

The basic MCDA-RL [10] defines two reference parameters: r_i=*reservation level*=the upper limit, not allowed to be crossed by the actual decision variable v_i of a solution; a_i=*aspiration level*=the lower bound beyond which the decision variables (and therefore, the associate solutions) are seen as similar (i.e., any solution can be seen as "good"-from the point of view of this variable). Applying these for each decision variable v_i, one can define two values named r_i and a_i, i= 1, ..m, by computing among all solutions s = 1, 2, ..S:

$$\begin{aligned} r_i &= \max [v_{is}],\ s = 1, 2, ..S \\ a_i &= \min [v_{is}],\ s = 1, 2, ..S \end{aligned} \quad (7)$$

An important modification is proposed in [16], aiming to make the algorithm agnostic versus different nature of criteria. The absolute value v_i of any decision variable is *replaced* with *distance from it to the reservation level*: r_i-v_i; (so, increasing v_i will decrease the distance); *normalization* is also introduced, in order *to get non-dimensional values, which can be numerically compared* despite their different nature. For each variable v_{si}, a ratio is computed:

$$v_{si}' = (r_i - v_{si})/(r_i - a_i), \quad \forall s, i \tag{8}$$

The factor $1/(r_i-a_i)$ - plays also the role of a weight. A variable for which the possible dispersion of values is high (max – min has a high value in formula (6)) will have lower weight and so, greater chances to be considered in determination of the minimum in the next relation (7). On the other side, if the values *min, max* are rather close to each other, then any solution could be enough "good", w.r.t. that respective decision variable.

The basic MCDA-RL algorithm steps are (see also [13]):
Step 0. Compute the matrix M$\{v_{si}'\}$, s=1…S, i=1…m
Step 1. Compute for each candidate solution s, the minimum among all its normalized variables v_{si}':

$$\min_s = \min\{v_{si}'\};\ i=1...m \tag{9}$$

Step 2. Select the best solution:

$$v_{opt} = \max\ \{\ \min_s\ \},\ s=1,..S \tag{10}$$

Formula (7) selects for each candidate solution s, the worst case, i.e., the closest solution to the reservation level (after searching among all decision variables). Then the formula (8) selects among the solutions, the best one, i.e., that one having the highest value of the normalized parameter. One can also finally, select more than one solution (quasi-optimum solutions in a given range). The network provider might want to apply different policies when deciding the controller placement; so, some decision variables could be more important than others. A simple modification of the algorithm can support a variety of provider policies. The new normalized decision variables will be:

$$v_{si}' = w_i(r_i - v_{si})/(r_i - a_i) \tag{11}$$

where $w_i \in (0,1]$ is a weight (priority), depending on policy considerations. Its value can significantly influence the final selection. A lower value of w_i represents actually a higher priority of that parameter in the selection process.

V. MCDA-BASED IMPLEMENTATION FOR SDN CONTROLLER PLACEMENT

A proof of concept simulation program (written in *Python* language [1] [13]) has been constructed by the authors, to validate the MCDA–RL based CPP problem and allocation of forwarders to controllers. The program has been extended in this study with reliability-related evaluation features. The simulation program uses the standard libraries and additionally the *NetworkX* and *matplotlib,* in order to create and manipulate the network graphs.

The simplifying assumptions (they could be also seen as limitations) of the model studied here, are: the network architecture is flat, i.e., no disjoint regions are defined; the network graph is undirected; any network node can be a forwarder but also can collocate a controller; when computing paths or distances, the metrics are additive; the number of controllers is predefined; the data traffic aspects and signaling interactions are not considered; the dynamic variation of the traffic in the data plane is not considered.

A. The MCDA basic model

The basic model considered in this paper, to solve the CPP and CSP problems has two working modes:

a. static mode: the input data are: network graph (overlay or physical), link costs/capacities, shortest path distances between nodes (e.g., computed with Dijkstra algorithm based on additive metric), desired number of controllers, the criteria (decision variables –these could be anyone, among those of Section III,l or others) for MCDA, and weights assigned to the decision variables).

Two working phases are defined:

(1) *Phase 1*:

1.1. *Compute all controller placements* C_1, C_2, …. (i.e., the set of candidate solutions). The number of placements is C_n^k (n= total number of network nodes; k= number of controllers).

1.3. Compute the *values of the normalized metrics for each possible controller placement (i.e., future MCDA candidate solution)*, by using specialized algorithms and metrics like those defined in Section III.

The Phase 1 phase has as outputs the set of candidate solutions (i.e., placement instances) and their associated values to fill the entries of the matrix M defined in Section IV. The Phase 1 computation could be time consuming; it depends on network size, but also on the number of criteria selected and the complexity to compute the metrics like in Section III., Such computations could be performed off-line [5]. For instance, in a real network, a master SDN controller having all these information could perform these computations. However, in a network exposing high dynamicity computing the Phase 1 in real time is a challenging issue.

(2) *Phase 2*: MCDA-RL: define r_i and a_i, for each decision variable; eliminate those candidates having parameter values out of range defined by r_i; assign – if wanted – convenient weights w_i for different decision variables; compute the normalized variables (formula (8)); run the MCDA Step 0, 1 and 2 of the (formulas (9) and (10)).

The Phase 2 provides the CPP solution.

The pseudocode of basic MCDA-based optimization processing is high level presented below:

```
Start
```

```
  //Initialize
    {Parse the arguments;
      Define the decision variables vi, i=1,..m, ∀i,
          vi ≥ 0;}
  //Build candidate solutions
       {Build the weighted graphs;
        Build candidate solutions;//S= the set of candidates
       }
  {Compute all shortest paths between pairs of
       network   nodes; // Dijkstra algorithm}
  {Normalize the decision variables; //(formula (8)
   Compute the matrix M{vsi'}, s=1…S, i=1…m;
   Compute for each candidate s, the min. among all  its normalized variables vsi';/formula (9)
   }
   Select the best solution; //formula (10)
Stop
```

b. dynamic mode : the semantic of the word *dynamic* here is the fact that some parameters are randomly generated i.e, not predefined. The initial input information is the total number of network nodes (not the complete graph) and desired number of controllers. The graph (which could be full-mesh or not) and costs of the links are randomly generated by the program.

B. Resilience-capable models

As shown in Section III, more realistic scenarios consider the possible occurrence of controller and/or network failures events. It is desired a resilient system i.e., able to recover (as much as possible) after failure events. The optimization process aims now to proactively find trade-off solutions to provide still a convenient behavior of the overall system in failure cases.

- *Backup controllers*

A simple static solution for assignment/mapping of the forwarders (this is CSP problem) to primary and backup controllers is presented below. For a given placement of the controllers, let it be C_p, the identities of nodes playing the role of controllers are known. The simplest assignment/mapping of forwarders to controllers is based on the shortest path (metric is average estimated latency forwarder-controller) to a controller. So, an algorithm will compute, for a given placement C_p, the distances from each F_i to each controller CT_1, CT_2, …CT_k and select the closest controller, let it be CT_m, as primary controller for F_i.

How to define the backup controllers? A natural solution (supposing that the total number of controllers is still k) will be to allow a forwarder to migrate from a failed primary controller to another backup/secondary controller, selected from the same set. This backup controller can be determined by the above algorithm, as the second one in the ordered list (using the shortest distance as criterion). This assignment should be performed for every possible placement C_i. If CPP optimization and forwarders-to-controllers assignment is wanted for the backup controllers, then it is necessary to add a new criterion (decision variable- e.g., similar to the average distance given by the formula (1)) to the MCDA algorithm, with a metric similar to that of formula (1). The reason is that for primary controller placement and forwarder assignment, one can find C_i as the best solution while for and backup controller placement and assignment other different C_j could be the best. Therefore, the MCDA can provide the best trade-off.

An auxiliary algorithm is used to compute a simple metric (average distance to a backup controller) to be added to MCDA. We introduce a novel decision variable *dist_backup* and perform the following computation (for each possible controller placement Ci containing the controllers CT_1, CT_2, ….CT_k):

```
For each forwarder Fi, i=1..N
 Do
   Dist_backup = 0;
   Compute dist. from Fi to any CTj, j=1..k;
   Dist_backup=Dist_backup + second_shortest_cost;
  Od
   Dist_backup_avg = Dist_backup/N;
```

This `Dist_backup_avg` can be added as a new decision variable to MCDA (maybe with appropriate wight) Therefore, the optimization will select a solution which considers also the backup controller placement and assignment of forwarder nodes as a factors influencing the final solution selection.

A simple example (Figure 1) will show the need of the additional MCDA criterion for the backup controllers. The example network is represented by an undirected graph, where the metric indicated on the edges can be the average latency between nodes (vertices).

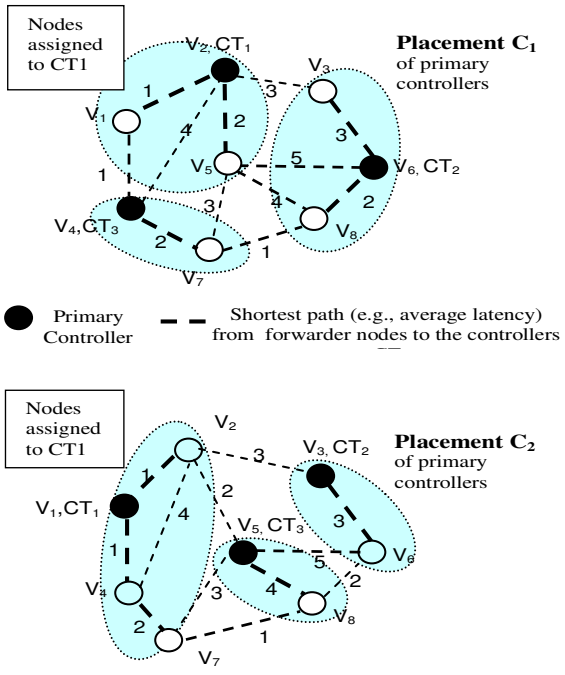


Figure 1. Simple example of two instances of primary controller placements and forwarders assignment

The network nodes are denoted with V_i, in order to emphasize that a given node can play the role of a forwarder but also a controller can be installed there. The edges costs correspond to an additive metric, i.e the average estimated communication delay between two nodes. The edges can represent real links or overlay ones (this aspect is irrelevant for the purposes of this study). It is assumed that three controllers exist, CT_1, CT_2, CT_3. We consider two placements of the *primary* controller placement C_1 and C_2. If one uses the optimization criterion given by the formula (1), then the assignment of forwarders to controllers are determined by the *shortest path* of each node to CT_1, CT_2, CT_3. The list of ordered distances can also provide the identity of the backup controller for each node V_i. Table I clarifies the assignment of the primary and backup controllers of the placement C_1. A similar table is valid for C_2, etc.

TABLE I. EXAMPLE: DISTANCES FROM NODES TO CONTROLLERS AND BACKUP CONTROLLERS DETERMINATION (PLACEMENT C_1)

				Selection	
Controller/ Node	**CT_1**	**CT_2**	**CT_3**	**Primary**	**Backup**
V_1	**1**	6	1	CT_1	CT_3
V_2	**0**	6	2	CT_1	CT_3
V_3	3	**3**	5	CT_2	CT_1
V_4	2	4	**0**	CT_3	CT_1
V_5	**2**	5	4	CT_1	CT_3
V_6	6	**0**	4	CT_2	CT_3
V_7	4	3	**2**	CT_3	CT_2
V_8	5	**2**	3	CT_2	CT_3

Considering the values of Table I the best assignment of forwarders to controllers for C_1 placement, is:
Primary controllers:
CT_1: {V_3, V_4}, CT_2: {V_3, V_6, V_8 }, CT_3: { V_4, V_7}
Backup controllers:
CT_1: {V_3, V_4}, CT_2: {V_7}, CT_3: { V_1, V_2, V_5, V_6, V_8 }
Analyzing the results two conclusions can be drawn:

- the assignment of the fowarders to primary controllers and respectively backup, can be very different
- the balance between solutions can be also very different; one can see the unbalance of the backup controller assignment.

Simple computations show that the average values of distances for the primary and respectively backup controllers are 1.25 and 2.75.

For another placement instance, i.e., C_2 (see Figure 1) one gets:
Primary controllers:
CT_1: {V_1, V_2, V_4, V_7}, CT_2: {V_3, V_6}, CT_3: { V_5, V_8}
Backup controllers:
CT_1: {V_3, V_5, V_8}, CT_2: {Ø}, CT_3: { V_1, V_2, V_4, V_6, V_7 }

The average values of distances for the primary and respectively backup controllers are 1.5 and 3.5. So one can say that C_1 placement is a better solution.

Even such simple examples prove the real need and usefulness of multi-criteria optimization, where resilience-oriented metrics can be added.

- *Load balancing for controllers*

As shown in Section III, a good balance of the node-to-controller distribution is desired as a proactive procedure to minimize the chance of future controller overload and to provide fairness between controllers. This paragraph will propose a simple load balancing solution for controllers. The solution is static, i.e., it will try to assign to different controllers, approximately, the same number of forwarders to be controlled. Note that such a solution will produce enough good results during the run-time, only if the data plane traffic distribution between the forwarders is rather uniform.

If the total number of nodes is N and the number of controllers is k, then the average number of nodes allocated to a controller is N/k. A simple new metric can be added to the set of MCDA criteria. This decision variable *D_avg* will measure the deviation of the actual number of nodes allocated to a controller CT_i, i.e., n_i, from the average value N/k, and averaging this for all controllers.

$$D\_avg = (1/N) \sum_{i=1\ldots k} |(n_i - N/k)| \quad (12)$$

If wanted, this variable can get an appropriate weight in the multi-criteria optimization process. If the example of the previous sub-section on backup controller problem is considered, then one can learn that solutions found there (based on latency criteria) could expose significant unbalance between controllers.

- *Nodes and link failures*

Nodes and link failures could appear in the network. Evaluation of effects of such events could be taken into account by adding new decision appropriate parameters in the set of MCDA input multi-criteria. Here, we adopted a different approach in comparison with the metric presented in Section III [7]. Given that most important metrics are forwarder-controller latency, inter-controller latency, load balancing of the controllers, optimization of the placement of the primary and backup controllers, the MCDA has been first run to produce controllers' placement optimization based on these important parameters. Then the simulation program allows some events to happen (e.g., nodes or link failures). The MCDA has been run again and produce a new placement after removing from the graph the failured entities. Finally, the placement produced in the updated conditions can be compared with the initial one, to evaluate if significant changes appeared. In such a way one can evaluate the robustness of the initial placement, and decide if that can be preserved or must be changed.

Two input parameters have been defined in the model:
nf- number of nodes supposed to fail
ef – number of links supposed to fail.

The specific nodes and links which will fail will be selected as to to simulate the "worst case", i.e., those nodes having the lowest cost of the adjacent links and, respectively those links having the least costs. If after second run of the MCDA, the initial placement of the controllers does not change, this means that initial placement has enough good robustness properties. Of course, this result will depend on selection of nf and ef values, for a given N nodes of the graph.

C. Simulation program for controller placement and selection optimization

The user interface of the simulation program (having resilience features included) is presented in Figure 2.

```
stefan@mint ~/Desktop/simulator_mcda $ python mcda.py -h
usage: mcda.py [-h] [-a [A]] [-w [W]] [-i [I]] [-b [B]] [-l [L]] [--dynamic] [-n N] [-c C] [-nf NF] [-ef
EF] [--debug]

Multi-criteria optimization algorithm
Optional arguments:
  -h, --help  show this help message and exit
  -a [A]      Average latency - failure free scenario. Expects a weight (priority) in interval (0, 1].
  -w [W]      Worst case latency - failure free scenario. Expects a weight (priority) in interval (0, 1].
  -i [I]      Inter controller latency. Expects a weight (priority) in interval (0, 1].
  -b [B]      Average latency - failure scenario. Expects a weight (priority) in interval (0, 1].
  -l [L]      Controller load-balancing. Expects a weight (priority) in interval (0, 1].
  --dynamic   Generate dynamic undirected graph
  -n N        Number of graph nodes. Valid only in dynamic mode.
  -c C        Number of controllers in graph. Valid only in dynamic mode.
              Allowed values are between N/3 and N/7
  -nf NF      Number of nodes that fail. Valid only in dynamic mode. Allowed  values: 1.. N-C.
  -ef EF      Number of edges that fail. Valid only in dynamic mode. Allowed  values: 1 ..N-C.
  --debug     Prints some computing results for debugging purposes.
```

Figure 2. The interface of the MCDA CPP simulation program

The decision parameters considered have been: *average* and *worst* latency between a forwarder and controller, *inter-controller* latency and *load balancing* related parameter. The program can be run in static or dynamic mode, with any number and set of criteria among those presented in the interface. The program is flexible in the sense that the set of decision weighted parameters (having appropriate metrics) can be enriched at will; the only needed modification is the number of columns of the matrix M.

Several numerical examples and results of the basic CPP solutions have been already presented in the work [13]. The current version of the implementation added reliability feature presented in Section IV.B.

The pseudo-code of the simulation program for dynamic mode is presented below, in high level view.

```
Start
   Generate the random graph;
   Generate all controlers' placements;
   Run MCDA;
   If link_failures are specified as a running
option then eliminate from the graph a number of
ef links having the minimum costs;
   If node_failures are specified as a running
option then eliminate from the graph a number of
nf nodes;
   If failures_are produced
      then {generate modified graph; Run MCDA;}
   Display the graphs;
Stop
```

D. Dynamic controller selection

In a dynamic network context, the controller selection (CSP) can be performed in a dynamic way. The multi-criteria algorithm can be as well applied in such cases. We consider here only the situations in which controller/node/link – failures occur.

In the static approach the backup controllers are predefined; their placement is selected by the optimization algorithm. For a real network, the algorithm can be run offline in a management center (in a hierarchical organization of the control plane, this could be a master SDN controller). This center is supposed to know all information in order to run MCDA-RL algorithm. The aspects related of collecting this information at the master SDN controller constitute a separate problem, which is not studied in this paper.

If a running forwarder looses its connectivity with its primary controller, it can act in two ways; a. try to connect to a known backup controller; b. select among several available controllers by running a MCDA algorithm. The input information for MCDA (decision criteria) could be:

- identities/addresses of a set of SDN controllers;
- degree of load for those controllers (e.g., periodically communicated, by a traffic monitoring system (having its central intelligence in the master SDN controller) to the forwarder
- local information observed by the forwarder, like connectivity to different nodes/controllers, etc. So, the forwarder can select based on MCDA-RL a novel controller.

VI. EXPERIMENTAL RESULTS

This section will shortly present some simple but relevant examples of results, in order to prove the validity of approach. The experiments are mainly oriented to test the resiliency.

- *Basic controller placement examples*

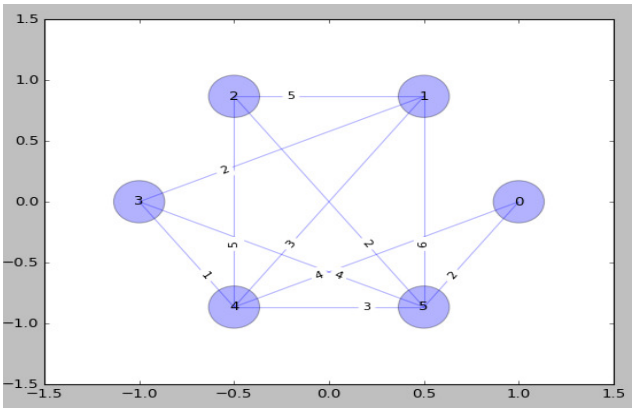


Figure 3. Static MCDA CPP optimization with individual criteria in MCDA

The objective of the first example is to show that applying a single optimization criteria the solutions can be very different. Figure 3 shows an example of optimization for a network statically defined, having N= 6 nodes and c= 2 controllers. Four controller placements have been considered:C_0: nodes [4, 5]; C_1: nodes [2, 4]; C_2: nodes [2, 5]; C_3: nodes [3, 5].

The result placements (examples) for different individual criteria are listed below:

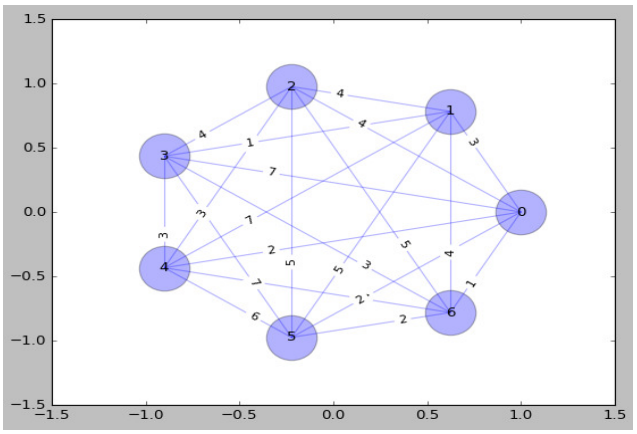
```
stephan@mint$ python mcda.py -a //average latency
to the primary controllers
Optimum Ci placement is Ci=3;
CT0 is placed in node 3
CT1 is placed in node 5
CT0 nodes {1,3,4}
CT1 nodes {0,2,5}
```

The computed latencies for the four placements are : C_0: 1.33; C_1: 1,66; C_2: 2,33; C_3: 1.13. One can see that C_3 is the best.

```
stephan@mint$ python mcda.py -b //average latency
to the backup controllers
Optimum Ci placement is Ci=0
CT0 is placed in node 4
CT1 is placed in node 5
CT0 nodes {1,3,4}
CT1 nodes {0,2,5}

stephan@mint$ python mcda.py -w //max latency to
the backup controllers
Optimum Ci placement is Ci=3
CT0 is placed in node 3
CT1 is placed in node 5
CT0 nodes {1,3,4}
CT1 nodes {0,2,5}

stephan@mint$ python mcda.py -i //inter-controller
latency
Optimum Ci placement is Ci=2
CT0 is placed in node 2
CT1 is placed in node 5
CT0 nodes {1,2}
CT1 nodes {0,3,4,5}
```



. Figure 4. Basic MCDA CPP optimization with dynamically generated network graph

The variety of the above results obtained for single criterion shows clearly the necessity of a multi-criteria optimization.

The second example shows a multi-criteria scenario. Figure 4 shows a graph dynamically generated with N=7 nodes and k=2 controllers. The optimization criteria have been *average latency*, *worst latency* and *inter-controller latency*, with equal weights $d_1=d_2=d_3=1$. The best placement selected is C2 , having the controllers placed in the nodes 0 and 3. The allocation of forwarders to controller can be selected based on shortest path principle. The command to run program and the main results are listed below.

```
stephan@mint$ python mcda.py -a 1 -w 1 -i 1 -
dynamic -n 7 -c 2
Optimum Ci placement is Ci=2
Controller is placed in node 0
Controller is placed in node 3
```

- *Load balancing for controllers*

Figure 5 shows an example in which the network graph has been dynamically generated with N=6 nodes and k= 2 controllers. The decision criteria have been *inter-controller latency* (weight = 1) and *balancing criterion* (weight = 0.5, i.e., having twice higher priority). The MCDA program has been run with parameters :

```
   stefan@mint$ python mcda.py -i 1 -l 0.5 --
dynamic -n 6 -c 2
```

The results obtained are: controllers CT_0 and CT_3 placed in the nodes 0 and 3. The allocation of forwarders to controllers are :

```
   Controller 0 has allocated node(s): 0, 2, 4.
   Controller 3 has allocated node(s): 1, 3, 5.
```

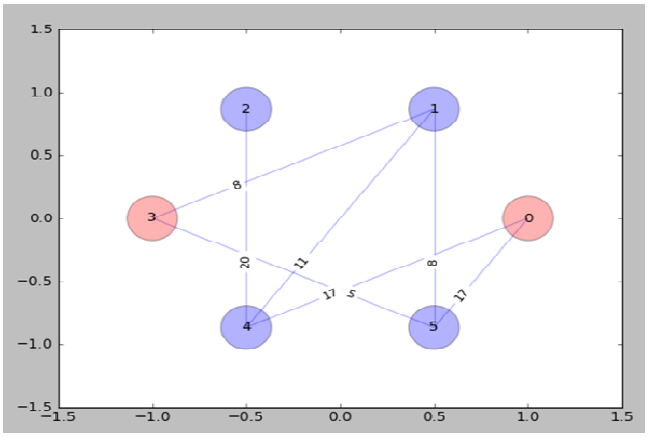


Figure 5. Example of a balanced allocation of the forwarders to controllers (after MCDA run)

Note that in this solution the inter-controller latency is taken into consideration, but the final value is not minimum; however, the allocation of the forwarders to controllers is balanced (3 forwarders per each controller). The reason is that load balancing criterion has been assigned a higher priority versus the inter-controller-latency.

- *Links and node failures*

To experiment such scenarios the simulator should be launched in *dynamic mode* and the number of links/nodes which will be in failure should be also specified. One can check if the placement selected is resilient to failures. For instance, if the unique parameter considered in MCDA would be the *average latency* of the *forwarders* to *backup* controllers, then one would expect that the resulting placement could be enough resilient to a low number of nodes and/or link failure events. Figure 6 shows such an example, by presenting the graphs resulted after running the program with the command:

```
python mcda.py -b --dynamic -n 8 -c 3 -ef 2
```

In this example, the network has N=8 nodes and c= 3 controllers; the number of failure links ef=2. This first placement (Figure 6a) has the controllers installed in nodes 3,4,5. The program is run again after some links failure (1-6, 3-7). Still the controller placement (i.e., after running again the MCDA on the reduced graph) is the same (Figure 6 – right), i.e., in the nodes 3,4,5.

Now we consider an experiment in which the criterion of the first run of the MCDA is to minimize the *average latency* between the *forwarders* and *primary* controllers (parameter introduced with weight = 1). The optimum placement of the controllers (with N=8, c=3), after first run of the MCDA, is in nodes 0, 2, 6. (Figure 7, left). Then two link failures are simulated (i.e., links 5-6 and 0-1 will be out of order). The command for such a run is:

```
python mcda.py -a --dynamic -n 8 -c 3 -ef 2
```

The optimum placement of the controllers in the new context (failure links) has been changed in nodes 3,5,6 (Figure 7b). So, one can conclude that the first placement is less resilient to link failures.

The lesson learned from such experiments is that there is no absolute unique optimum solution of such problems, to satisfy all requirements. Depending on the particular context of the SDN-controlled network and some network owner policies, different placement solutions can be found as more appropriate, to satisfy in a convenient way several criteria.

These examples illustrate the power of the MCDA algorithm where various sets of criteria and different priorities (driven by policies) can be considered.

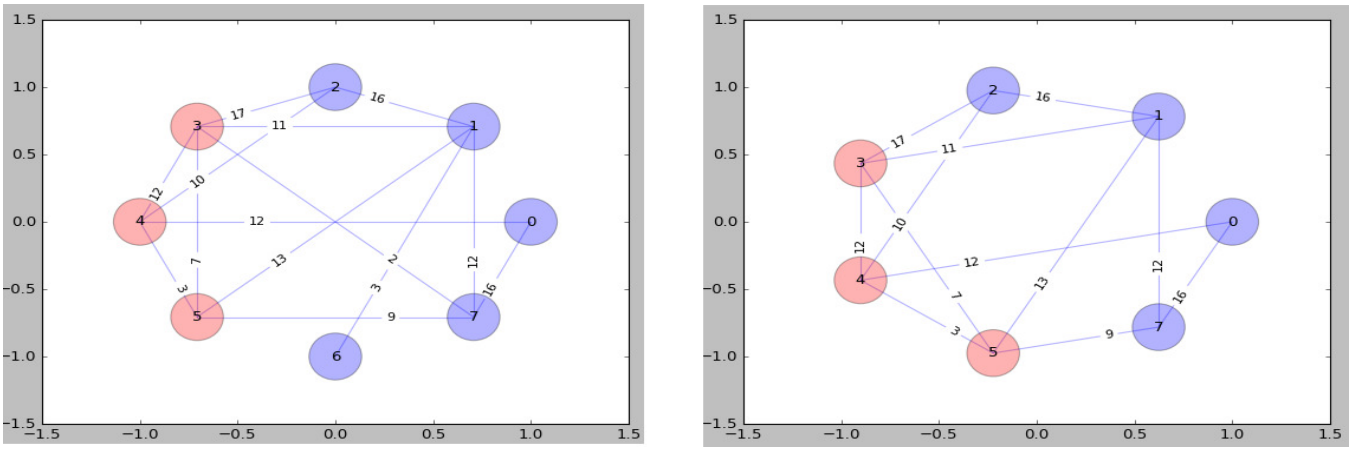


Figure 6. Example of controller placement resilient to link failures a.Left: placement before link failures; b.Right: placement after some links failures.

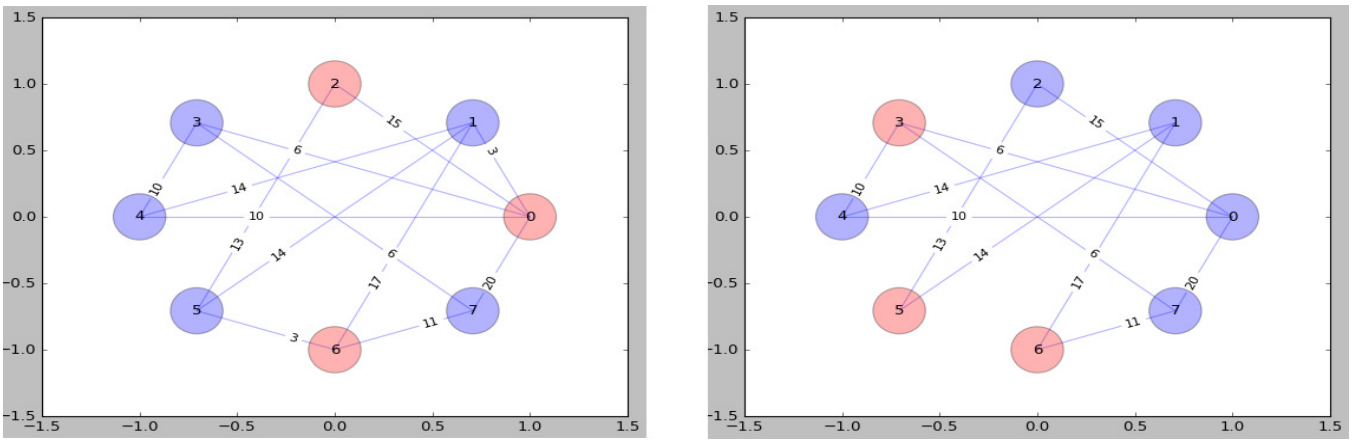


Figure 7. Example of placement non-resilient to link failures
a.Left: controller placement before link failures; b. Right: controller placement after some links failures (5-6, 0-1).

VII. CONCLUSIONS AND FUTURE WORK

This paper extended the study [1], on using multi-criteria decision algorithms (MCDA) to optimally place the controllers in large SDN, based networks, while aiming to achieve good resiliency properties of the system. It is illustrated the main MCDA advantage, i.e., that it can produce a tradeoff (optimum) result, while considering several weighted criteria, part of them even being partially contradictory.

This study provides (in comparison to [1]) more comprehensive discussion and analysis of resiliency-oriented properties of a SDN network with distributed control plane. Simple but relevant examples have been added, to show that actually no unique solution exists for controller placement to be optimal with respect to all criteria envisaged. Therefore, in practice, the weights of the decision parameters introduced in MCDA should be cleverly adopted, to meet the prioritized list of the network provider requirements.

This study has shown that actually the MCDA – based optimization can be performed in a flexible way:

- introducing in MCDA all decision parameters, with appropriate weights in order to achieve a trade-off solution after a single MCDA run;
- using iteratively several rounds (see Section VI), i.e., introducing first the most important parameters and run MCDA; then modify the topology/conditions and check if the first controller placement is still good enough in these new conditions; if not, then add parameters to MCDA and run again the algorithm.

The paper added several additional experimental results in Section VI. The forwarder-controller mapping optimization and backup controller selection have been also considered.

Future work will be still necessary for CPP and CSP problems. Experiments on large networks [22][23] could better validate the optimization solutions in a more realistic environment. Another important aspect can be the dynamic of the overall system during run-time, when the traffic amount inside different regions of the the data plane (i.e., between different forwarders) might have significant variations. This can lead to overload of some SDN controllers, especially if reactive-mode of the control plane is applied in those networks and given the limited controller processing capacity. This problem could be solved in two ways: a. moving some controllers (so, the placement will be modified) to the overloaded regions to better serve the forwarder requests for flow table configuration); b. dynamically migrate some switches/forwarders between the controllers, in order to better balance the controllers' load. For instance, in recent studies [20][21], the dynamic switch migration is optimized based on input information periodically provided by a monitoring system. However, these studies do not consider multi-criteria approach, but only the traffic load of the network data plane and impact on controller tasks. Here, combining MCDA with such traffic-based algorithms could provide better results.

REFERENCES

[1] E. Borcoci and S. Ghita, "Reliability-aware Optimization of the Controller Placement and Selection in SDN Large Area Networks", The Thirteenth International Conference on Systems and Networks Communications, ICSNC, Nice, 2018, https://www.iaria.org/conferences2018/ICSNC18.html, [retrieved: 6, 2019].

[2] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On Scalability of Software-Defined Networking", IEEE Comm. Magazine, pp. 136-141, February 2013.

[3] A. Tootoonchian and Y. Ganjali, "Hyperflow: a distributed control plane for openflow" in Proc. INM/WREN, 2010, https://pdfs.semanticscholar.org/f7bd/dc08b9d9e2993b363972b89e08e67dd8518b.pdf, [retrieved: 5, 2018].

[4] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, et al., "Onix: a distributed control platform for large-scale production networks," in Proc. OSDI, 2010, https://www.usenix.org/legacy/event/osdi10/tech/full_papers/Koponen.pdf, [retrieved: 5, 2018].

[5] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network Function Virtualisation: Challenges and Opportunities for

Innovations", IEEE Communications Magazine, pp. 90-97, February 2015.

[6] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in Proc. HotSDN, pp. 7–12, 2012, https://dl.acm.org/citation.cfm?id=2342444, [retrieved: 5, 2019].

[7] K. Sood and Y. Xiang, "The controller placement problem or the controller selection problem?", Journal of Communications and Information Networks, Vol.2, No.3, pp.1-9, Sept.2017.

[8] D. Hochba, "Approximation algorithms for np-hard problems", ACM SIGACT News, 28(2), pp. 40–52, 1997.

[9] E. Borcoci, T. Ambarus, and M. Vochin, „Multi-criteria based Optimization of Placement for Software Defined Networking Controllers and Forwarding Nodes," The 15th International Conference on Networks, ICN 2016, Lisbon, http://www.iaria.org/conferences2016/ICN16.html, [retrieved: 5, 2019].

[10] A. P. Wierzbicki, "The use of reference objectives in multiobjective optimization". Lecture Notes in Economics and Mathematical Systems, vol. 177. Springer-Verlag, pp. 468–486.

[11] S. Yoon, Z. Khalib1, N. Yaakob, and A. Amir, "Controller Placement Algorithms in Software Defined Network - A Review of Trends and Challenges", MATEC Web of Conferences ICEESI 2017 140, 01014 DOI:10.1051/matecconf/201714001014, 2017.

[12] G.Wang, Y.Zhao, J.Huang, and W.Wang, "The Controller Placement Problem in Software Defined Networking: A Survey", IEEE Network, pp. 21- 27, September/October 2017.

[13] A. K. Singh and S. Srivastava, "A survey and classification of controller placement problem in SDN", International Journal of Network Management, March 2018, pp 1-25, https://doi.org/10.1002/nem.2018.

[14] A.Kumari and A.S.Sairam, "A Survey of Controller Placement Problem in Software Defined Networks", arXiv:1905.04649v1 [cs.NI] 12 May 2019.

[15] H. Yan-nan, W. Wen-dong, G. Xiang-yang, Q. Xi-rong, and C. Shi-duan, "On the placement of controllers in software-defined networks", ELSEVIER, Science Direct, vol. 19, Suppl.2, pp. 92–97, October 2012, , http://www.sciencedirect.com/science/article/pii/S10058851160438X, [retrieved: 1, 2018]..

[16] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia, "Pareto-Optimal Resilient Controller Placement in SDN-based Core Networks," Proceedings of the ITC, Shanghai, China, 2013, https://ieeexplore.ieee.org/document/6662939/, [retrieved: 1, 2019].

[17] Y. Hu, W. Wendong, X. Gong, X. Que, and C. Shiduan, "Reliability aware controller placement for software-defined networks," in Proc. IM. IEEE, pp. 672–675, 2013, https://ieeexplore.ieee.org/document/6573050/, [retrieved: 1, 2019].

[18] L. Muller, R. Oliveira, M. Luizelli, L. Gaspary, and M. Barcellos, "Survivor: an Enhanced Controller Placement Strategy for Improving SDN Survivability", IEEE Global Comm. Conference (GLOBECOM); 12/2014, https://ieeexplore.ieee.org/document/7037087/, [retrieved: 4, 2018].

[19] Y. Zhang, N. Beheshti, and M. Tatipamula, "On Resilience of Split-Architecture Networks" in GLOBECOM 2011, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.691.795&rep=rep1&type=pdf, [retrieved: 5, 2019]. .

[20] M. Cello, Y. Xu, A. Walid, G. Wilfong, H. J. Chao, et al., "Balcon: A distributed elastic SDN control via efficient switch migration", in Proc. IEEE Int. Conf. Cloud Eng. (IC2E), April 2017, pp. 40–50.

[21] Y.Xu, M.Cello, I-Chih Wang, A. Walid, G. Wilfong, et al., "Dynamic Switch Migration in Distributed Software-Defined Networks to Achieve Controller Load Balance", IEEE Journal on Selected Areas in Communications , Vol. 37, No. 3, March 2019, pp. 515-528.

[22] Internet2 open science, scholarship and services exchange. http://www.internet2.edu/network/ose/, [retrieved: 4, 2018].

[23] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet Topology Zoo," IEEE JSAC, vol. 29, no. 9, 2011, pp.1765-1475.

[24] F. Al-Tam and N.Correia, "On Load Balancing via Switch Migration in Software-Defined Networking", IEEE Access, DOI 10.1109/ACCESS.2019.2929651, pp.1-13, July 2019.