# Robust SDN Controller Placement to Malicious Node Attacks

Dorabella Santos
Instituto de Telecomunicações
Aveiro, Portugal
dorabella@av.it.pt

Amaro de Sousa
Instituto de Telecomunicações
DETI, Universidade de Aveiro
Aveiro, Portugal
asou@ua.pt

Carmen Mas Machuca
Technical University of Munich
Chair of Communication Networks
Munich, Germany
cmas@tum.de

***Abstract*—In software-defined networking (SDN), the control plane is separated from the switching plane (*i.e.*, data plane). The logically centralized control plane is implemented by physically distributing several controllers throughout the network for scalability and resilience. The problem of finding the number and location of the SDN controllers is known as the controller placement problem (CPP). In this paper, we consider the following robust CPP problem variant. For a given maximum switch-controller (SC) delay and a given maximum controller-controller (CC) delay in the regular state, we aim to find a CPP solution that maximizes the network robustness for a given number of malicious node attacks. First, we guarantee that if all but one controller nodes are shutdown, there is still a switching path from any switch to the surviving controller. We propose an ILP based method aiming to enumerate all such solutions. Then, for different malicious node attacks corresponding to different attacker's strategies, we evaluate the previous solutions to determine the ones that maximize the minimum number of switches that can still be connected to at least one controller. We compare the robust CPP solutions with non-robust CPP solutions which aim solely to minimize the average SC delay or average CC delay. In the latter cases, we propose ILP models that can be solved efficiently by standard solvers. Finally, we present a set of computational results showing the trade-off between the robustness improvement of the proposed solutions against the resulting penalties on the SC and CC delays.**

***Keywords—SDN, controller placement, malicious node attacks, integer linear programming***

## I. INTRODUCTION

Disaster based failures can seriously disrupt a telecommunications network, making its services unavailable [1]. Due to the current importance of telecommunication services and the rising risk of large human-made attacks, improving the preparedness of networks to such attacks is becoming a key issue (see [2] for a recent survey on security challenges in communication networks conducted within the COST Action RECODIS). In this paper, we consider disasters caused by malicious attacks to multiple network nodes. When such a disaster occurs, it is important not only to quickly recover the shutdown nodes but also to minimize its impact on the capacity of the surviving nodes to keep working properly.

Software-defined networking (SDN) is a network architecture that separates the control plane from the switching plane (*i.e.*, data plane). The logically centralized control plane is implemented by placing controllers at different locations for scalability and resilience. How many controllers and where to place them is commonly known as the controller placement problem (CPP), a facility location type of problem known to be NP-hard [3]. Note that SDN exhibits unique vulnerabilities to malicious node attacks. For example, if an attacker is able to shut down all controller places, the whole network collapses even if the surviving network is fully connected.

In general, the network can be either in the regular state (*i.e.*, with all nodes operational) or in a failure state (*i.e.*, when one or more nodes are shutdown). We assume that the operator aims to improve the network robustness to malicious attacks of $p$ simultaneous nodes and, therefore, the operator aims to distribute $C$ controllers with $C = p + 1$ (if $C$ is less, an attacker knowing the controller locations may be able to shut down all controllers causing total network collapse).

In this paper, we address the following robust variant of the CPP problem. The CPP solution must (i) be compliant with a given maximum switch-controller (SC) delay and a given maximum controller-controller (CC) delay in the regular state and (ii) guarantee in the failure state that if any $p$ controller nodes are shutdown, there is still a switching path from any switch to the surviving controller. Then, among all solutions fulfilling these requirements (i.e., all feasible solutions), the aim is to find the CPP solutions that maximize the network robustness for different malicious node attacks corresponding to different possible attacker's strategies. The optimal CPP solutions are the ones that maximize the minimum number of switches that can still be connected to at least one controller node among all attacks. To compute the robust CPP solutions, we propose an integer linear programming (ILP) based methodology, and a speedup technique, aiming to enumerate all feasible solutions and, then, we evaluate each solution to select the optimal ones. We also compare the robust CPP solutions with non-robust CPP solutions which aim solely to minimize the average SC and CC delays. Finally, we present computational results showing the trade-off between the robustness improvement of the proposed solution against the resulting penalties on the SC and CC delays.

In general, the CPP solution is biased by the considered constraints. The most common constraint is related to the SC delay. Until recently, most of the literature has focused in

minimizing the average or the maximum SC delays in order to reduce the time to set up new flows requested by a switch [4]. The authors in [5] propose a CPP solution minimizing the number of required controllers and SC delay. However, the distribution of different controllers supported by e.g., ONOS [6] and ODL [7], adds requirements to guarantee an efficient synchronization among controllers and to keep a consistent network view. These requirements are related to the distance between controllers. The most used metric for this purpose is the maximum CC delay [5]. Note that (i) minimizing the average SC delay, in general, increases the average CC delay, and vice-versa, as shown in [8], and (ii) increasing the number of controllers will, in general, decreases the average SC delay but increases the average CC delay (controllers placed closer to the switches become more spread over the whole network).

In order to overcome the limitation of having each switch assigned to one controller through one control channel, as initially considered, several works have proposed CPP solutions to make SDN more resilient to failures. In [9], the control channel availability is increased by considering (i) two disjoint control paths between any switch and its assigned controller or (ii) two controller replicas to each switch with disjoint control paths. The solutions consider SC delays and show that both strategies provide significant gains in control path availability, while adding a limited penalty to the average SC delay in case of single failures. In [10], the authors address controller failures presenting ILP models for the regular and failure states. For the regular state, the goal is to minimize the number of controllers assuming maximum values for both SC and CC delays to guarantee reasonable network performance (they assume that each switch connects to the closest available controller). Load balance is also guaranteed by imposing a given maximum value to the load difference among controllers. For the failure state, the controllers are assumed to fail with a given probability and, when a controller does fail, each of its switches reconnects to the closest available controller. The objective in this case is a combination of minimizing the number of controllers and the average SC delays.

Getting closer to the multiple failure scenario considered in this paper, the authors in [11] address targeted attacks to an SDN network. Assuming that the attacker has knowledge of the network topology but neither the number of controllers nor their location, the authors study the network vulnerabilities to centrality-based attacks. The proposed algorithm proposes controller placements based on the least critical nodes, as a way to cheat the attacker (*i.e.*, the attacker is expected to choose the most critical nodes hoping to cause maximum disruption). On the other hand, authors in [12] propose a CPP solution that, given a multiple failure scenario, finds the location of the controllers given the min cut sets of the topology.

This paper is organized as follows. Section II discusses how non-robust CPP solutions can be optimally computed to minimize the average SC and CC delays. Section III explains how to compute the robust CPP solutions and describes the set of considered malicious node attacks. Section IV presents a set of computational results showing the trade-off between the SDN robustness improvement and the resulting penalties on the average SC and CC delays. Finally, Section V presents the main conclusions of the work.

## II. The Non-Robust CPP Problem

Consider a SDN switching network represented by a directed graph $G = (N, A)$, where $N$ is the set of SDN switches and $A$ is the set of directed links. The total number of nodes is $|N| = n$ and each link direction is represented by the arc $(i, j)$. Also consider $V(i)$ as the set of neighbor nodes of node $i$. Given the delay of each arc, the shortest path delay between nodes $i$ and $j$ is denoted as $d_{ij}$.

In the non-robust CPP problem, the aim is to optimize the SDN control plane performance in its regular state by minimizing the average SC and CC delays and guaranteeing that: (i) the SC delay between any switch and its primary controller does not exceed a given $D_{sc}$ and (ii) the CC delay between any pair of controllers does not exceed a given $D_{cc}$. Consider the following decision variables:

$y_i \in \{0,1\}$ binary variable that is 1 if a controller is placed in node $i$, and 0 otherwise.

$z_{ij} \in \{0,1\}$ binary variable that is 1 if the primary controller of switch $i$ is placed in node $j$, and 0 otherwise.

$c_{ij} \in \{0,1\}$ binary variable that is 1 if a controller is placed on node $i$ and another controller is placed on node $j$, with $j > i$, and 0 otherwise (*i.e.*, $c_{ij} = y_i \cdot y_j$).

Then, a proper set of linear constraints defining the set of all feasible solutions is as follows:

$$\sum_{i \in N} y_i = C \tag{1}$$

$$\sum_{j: d_{ij} \le D_{sc}} y_j \ge 1 \qquad i \in N \tag{2}$$

$$y_i + y_j \le 1 \qquad i \in N, j \in N: d_{ij} > D_{cc} \tag{3}$$

$$\sum_{j: d_{ij} \le D_{sc}} z_{ij} = 1 \qquad i \in N \tag{4}$$

$$z_{ij} \le y_j \qquad i \in N, j \in N \tag{5}$$

$$c_{ij} \le y_i \qquad i \in N, j \in N: j > i \tag{6}$$

$$c_{ij} \le y_j \qquad i \in N, j \in N: j > i \tag{7}$$

$$c_{ij} \ge y_i + y_j - 1 \qquad i \in N, j \in N: j > i \tag{8}$$

$$y_i \in \{0,1\} \qquad i \in N \tag{9}$$

$$z_{ij} \in \{0,1\} \qquad i \in N, j \in N \tag{10}$$

$$c_{ij} \in \{0,1\} \qquad i \in N, j \in N: j > i \tag{11}$$

Constraints (1–3) are the basic set of constraints: constraint (1) guarantees that $C$ controller nodes are selected; for each node $i$, constraints (2) guarantee that there is at least one controller in some node distanced at most $D_{sc}$ from $i$ (including itself) and constraints (3) guarantee that any two controllers are not placed in nodes distanced further than $D_{cc}$ from each other. Constraints (4–5) guarantee a proper assignment of primary controllers to switches: constraints (4) guarantee that the primary controller of each switch is placed in a node whose distance in not higher than $D_{sc}$ and constraints (5) guarantee that if the primary controller of a switch is at node $j$, a controller must be placed at that node. Constraints (6–8) guarantee a proper assignment of variables $c_{ij}$ (they are the

standard set of linear constraints imposing $c_{ij} = y_i \cdot y_j$). Finally, constraints (9–10) are the variable domain constraints.

The average SC delay of a feasible solution is the sum of all SC delays divided by the number of switches without collocated controllers:

$$f_{sc}(z) = \frac{1}{n-C} \sum_{i \in N} \sum_{j \in N \setminus \{i\}} d_{ij} z_{ij}$$

The average CC delay of a feasible solution is the sum of all CC delays divided by the number of controller pairs:

$$f_{cc}(c) = \frac{2}{C(C-1)} \sum_{i \in N} \sum_{j \in N: j > i} d_{ij} c_{ij}$$

It is well-known (as in [8]) that, in general, there is no single solution that simultaneously minimizes both functions $f_{sc}(z)$ and $f_{cc}(c)$. The joint optimization of both functions is a bi-objective optimization problem and has multiple optimal solutions (the so-called Pareto solutions) representing different trade-offs between the two objectives. Here, we consider the two extreme cases:

*MinAvgSC* solution – the solution that minimizes the average SC delay and, among all such solutions, the one that minimizes the average CC delay.

*MinAvgCC* solution – the solution that minimizes the average CC delay and, among all such solutions, the one that minimizes the average SC delay.

These solutions can be determined by solving in sequence two ILP models. To compute the *MinAvgSC* solution, we first solve the model:

Minimize $f_{sc}(z)$

Subject to: (1–11)

and then, assuming its optimal value $z_{sc}$, we solve the model:

Minimize $f_{cc}(c)$

Subject to: $f_{sc}(z) \leq z_{sc}$ , (1–11)

The solution of the second ILP model minimizes the average CC delay while guaranteeing that the minimum average SC delay $z_{sc}$ is met. The *MinAvgCC* solution is computed similarly with $f_{cc}(c)$ as the objective of the first model and $f_{sc}(z)$ as the objective of the second model. In our computational results, both methods are solved very efficiently by a standard ILP solver (we used CPLEX 12.6.1) where the total runtime was always below 6 seconds in all cases.

## III. The Robust CPP Problem

Consider a SDN switching network modelled as described at the beginning of Section II. In a failure state, we assume that each switch dynamically selects the closest surviving controller as its primary controller and, so, any controller acts as a backup controller for any switch. Moreover, we assume that when a node hosting a controller is shutdown, both the controller and its collocated switch fail.

In the robust CPP problem, besides the maximum SC and CC delay requirements related with the regular state, the selection of the $C$ controller nodes must satisfy an additional requirement: there must be a routing path from each switch to each controller not passing through any other controller. This requirement guarantees that all switches can still connect to the surviving controller if any $p = C - 1$ controller nodes are shutdown (although, in the failure state, the maximum delay requirement imposed for the regular state might be not ensured). Consider variables $y_i$ as defined in Section II (variables $z_{ij}$ and $c_{ij}$ are no longer needed) and the following new variables:

$x_{ij}^k \in \mathbb{N}_0^+$ non-negative integer variable indicating the number of paths that use arc $(i,j)$ from switch $k$ to all controllers.

A proper set of linear constraints defining the set of all feasible solutions of the robust CPP problem is as follows:

(1–3), (9)

$$\sum_{j \in V(i)} \left(x_{ij}^k - x_{ji}^k\right) \leq y_i \qquad k \in N, i \in N \setminus \{k\} \qquad (12)$$

$$\sum_{j \in V(i)} \left(x_{ij}^k - x_{ji}^k\right) \geq 0 \qquad k \in N, i \in N \setminus \{k\} \qquad (13)$$

$$\sum_{j \in V(i)} x_{ij}^k \leq C(1 - y_i) \qquad k \in N, i \in N \qquad (14)$$

$$\sum_{j \in V(i)} x_{ji}^k \geq y_i - y_k \qquad k \in N, i \in N \setminus \{k\} \qquad (15)$$

$$x_{ik}^k = 0 \qquad k \in N, i \in V(k) \qquad (16)$$

$$x_{ij}^k \in \mathbb{N}_0^+ \qquad k \in N, (i,j) \in A \qquad (17)$$

Constraints (1–3) and (9) guarantee that variables $y_i$ define a proper set of $C$ controller places such that both maximum SC and CC delays are guaranteed (as explained in Section II).

Constraints (12–17) guarantee the additional requirement in the following way. For each switch $k \in N$, if node $i$ does not host a controller (*i.e.* $y_i = 0$), constraints (12–13) are equivalent to $\sum_{j \in V(i)} \left(x_{ij}^k - x_{ji}^k\right) = 0$ and are the typical path conservation constraints; in this case, constraints (14–15) are redundant. On the other hand, if node $i$ hosts a controller *(i.e.*, $y_i = 1$), constraints (14) guarantee that there are no node $i$ outgoing arcs in any path from $k$ (ensuring that all paths originated at $k$ have no intermediate controller nodes) and constraints (12–13) and (15) guarantee that there is exactly one path ending at node $i$ (ensuring that there is one path originated at $k$ that reaches each controller). Finally, constraints (16) guarantee that there is no path from $k$ to itself and constraints (17) are the domain constraints of the new variables $x_{ij}^k$.

In order to understand constraints (12–17), let us consider the example presented in Fig. 1, which depicts an SDN network with 8 switching nodes and $C$ = 3 controllers (controller locations highlighted in gray). The controller locations of Fig.1(a) are eliminated by constraints (12–17) since there is no path from node 7 to controller placed at node 2 without passing either through node 5 or 6 (the same happens with node 8). In this case, if controller nodes 5 and 6 are shut down, the surviving network is not fully operational. On the other hand, constraints (12–17) allow the controller locations of Fig. 1(b) since, from every switching node there is always a routing path to each controller not passing through any other

controller. Fig. 1(b) shows a set of $x_{ij}^{k}$ variable values (the variables not shown are 0) that are compliant with constraints (12–17) and define a set of three paths from node 1: one to node 3, one to node 4 and one to node 7. In this case, if any two controller locations are shutdown, the surviving network is still fully operational.

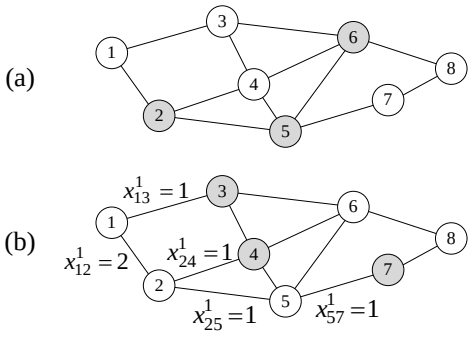


Fig. 1. Illustration of the additional property

The final goal of the robust CPP problem is to find the feasible solution that maximizes the network robustness for a given set of malicious node attacks. The most standard way to reach this goal would be to define a proper ILP model whose optimal solution is the targeted CPP solution. In this work, however, we adopt a different approach. First, we aim to obtain all feasible solutions, *i.e.*, the solutions defined by (1–3), (9), (12–17), or at least a large number of them (when the total set is too large). Then, we evaluate each solution against a given objective (or a combination of objectives). Since this evaluation runs in polynomial time, we have total flexibility in the way we can define the target optimization goal.

In the next three subsections, we separately describe, first, a basic method to compute the feasible solutions, then, a speedup technique that, plugged into the basic method, reduces significantly its computational runtimes and, lastly, how all solutions are computed to find the most robust solutions for a given set of malicious node attacks.

### *A. Basic Method*

To obtain all possible solutions, we define the following ILP model:

Maximize $\sum_{i\in N} \varphi_i y_i$ (18)

Subject to: (1–3), (9), (12–17)

The coefficients $\varphi_i$ of the objective function (18) are given by the values of a node centrality measure. This is motivated by the assumption that nodes with higher centrality values are more promising candidates to place controllers (in our computational results we have used the closeness centrality).

To obtain all feasible solutions, we start by solving the above ILP model which returns the first solution (defining a set of *C* controller nodes). Then, iteratively, the obtained set of controller nodes $\{\rho_1, \dots, \rho_C\}$ of the previous solution is eliminated from the feasible set by adding constraint $y_{\rho_1} + \cdots + y_{\rho_C} \leq C - 1$ to the ILP model and the augmented model is solved. The iterations continue until the augmented model is infeasible, meaning that all solutions have been found. When the number of feasible solutions is very high, this method can take too long. So, we consider a $L_{\max}$ parameter and we stop the procedure when either the model is infeasible or $L_{\max}$ solutions have been generated.

### *B. Speedup Technique*

In practice, solving an ILP model for each feasible solution can be very time consuming. To reduce the required runtime, we devised a speedup technique where most of the feasible solutions are computed by a more efficient means.

The speedup technique is based on a random walk that starts with a feasible solution and repeatedly moves from a feasible solution to a neighbor solution by randomly changing the location of one controller. If the neighbor solution is feasible, the random walk progresses from it; otherwise, the procedure progresses from the previous feasible solution. All feasible solutions are stored and the random walk stops when $I_{\max}$ consecutive neighbor solutions are infeasible ($I_{\max}$ is an input parameter). The plugging of the speedup technique into the basic method is as follows:

(a) A feasible solution is computed by solving the ILP model. If the ILP model is infeasible, the procedure stops.

(b) The solution of (a) is used to generate a neighbor solution, by randomly changing a controller from its current node to a neighboring node that does not already have a controller – we refer to this change as a controller hopping operation.

(c) The neighbor solution is evaluated for feasibility. If feasible, the neighbor solution becomes the current solution and the controller hopping operation is applied to generate a neighbor solution.

(d) Step (c) is repeated until the neighbor solution is infeasible; then, the procedure sets the current solution to the last feasible solution and the controller hopping operation is applied to generate a neighbor solution.

(e) The generation of neighbor solutions continues until a maximum number $I_{\max}$ of infeasible solutions are consecutively generated. Then, the ILP model is augmented with the constraints associated to all feasible solutions meanwhile computed and the procedure returns to step (a).

In addition, a list of feasible solutions is initialized empty and a feasible solution counter is initialized as0. Then, in all steps where a new feasible solution is computed, and if it is not in the list: (i) the solution is added to the list, (ii) the feasible solution counter is incremented and (iii) the procedure stops if the counter reaches the value $L_{\max}$.

Note that the feasibility evaluation of a neighbor solution has polynomial complexity. Recall that a solution is feasible if it is compliant with the maximum given $D_{sc}$ and $D_{cc}$ delays in the regular state and if the controller locations are compliant with the additional requirement. First, the shortest path delay $d_{ij}$ between all pairs of nodes can be computed in polynomial time and only once at the beginning. To check if the maximum SC delay is met, we need to compute for each node *i* not hosting a controller the minimum value $d_{ij}$ for all nodes *j* hosting a controller, which has complexity $\Theta(n \times C)$. To check if the maximum CC delay is met, we need to compute the

minimum value $d_{ij}$ among all node pairs $i$ and $j$ hosting a controller each, which has complexity $\Theta(C \times C)$. Finally, to check if there is a routing path from each switch to each controller not passing through any other controller, we first eliminate from the original graph $G = (N, A)$ all arcs $(i, j)$ for all nodes $i$ that host a controller. Then, in this graph and for each node $i$ not hosting a controller, we need to compute the shortest path tree from $i$ to all nodes hosting controllers, which has complexity $\Theta(n^2)$.

### C. Robust CPP Solutions

With the feasible solutions computed as described in the previous subsections, the aim is to find those that maximize the network robustness to a given set of $M$ malicious attacks to $p$ simultaneous nodes. Each malicious attack is defined by a set of $p$ nodes that the attacker can shut down simultaneously. A set of $M$ malicious attacks is a set of $M$ different combinations of $p$ nodes such that the nodes of each combination can be simultaneously shutdown.

The robustness of each feasible solution is evaluated as follows. For each combination $m = 1, \ldots, M$ of $p$ nodes, we start by computing a graph $G_m$ by eliminating from the original graph $G = (N, A)$ all $p$ nodes. Then, we compute the number $n_s^m$ of surviving nodes that have connectivity to at least one surviving controller node in graph $G_m$. In this process, we also compute the number $n_{sc}^m$ of surviving nodes whose shortest path delay to its primary controller is not higher than $D_{sc}$. Finally, $n_s$ is computed as the minimum among all $n_s^m$ values and $n_{sc}$ is computed as the minimum among all $n_{sc}^m$ values. Besides the graph transformations, the robustness evaluation of each feasible solution has complexity $\Theta(M \times n^2)$. For each feasible solution, $n_s$ is the minimum number of nodes that still have a primary controller among all $M$ attacks and $n_{sc}$ is the minimum number of nodes that still have a primary controller with a SC delay not higher than $D_{sc}$ (the maximum delay required for the regular state) among all $M$ attacks. Then, when comparing the robustness of different feasible solutions, we consider that a solution is better if (i) it has a higher value of $n_{sc}$ or (ii) the same value of $n_{sc}$ and a higher value of $n_s$.

In order to define a proper set of malicious node attacks, we have assumed, as in other works (for example [11]), that the attacker has knowledge of the network topology but neither of the number of controllers or their location. In this case, the attacker chooses the $p$ nodes based on node centrality metrics. We have considered $M = 3$ attacks: the $p$ nodes are selected by node degree (measures how many direct connections the node has with other nodes), node closeness (measures how close each node is to all other nodes) and node betweenness (measures how many shortest paths between all other node pairs include each node). In all cases, the node with the highest centrality value is first selected. Then, the selected node is eliminated from the graph, the node centrality values are recomputed and the resulting highest centrality node is selected. The selection continues until $p$ nodes are selected.

## IV. Computational Results

In the computational results, we have considered two network topologies. Germany50 (Fig. 2) has 50 nodes, 88 undirected links and an average node degree of 3.52 (information available at http://sndlib.zib.de). CORONET CONUS (Fig. 3) has 75 nodes, 99 undirected links and an average node degree of 2.64 (information available at http://www.monarchna.com/topology.html). As in other works (for example [10]), we have defined the delays in terms of shortest path lengths and the maximum delay parameters ($D_{sc}$ and $D_{cc}$) are defined as percentages of the graph diameter $D_g$ (the largest shortest path length among all node pairs). Based on the node geographical coordinates of each network topology, we have determined the length of each link by computing the shortest path length over the Earth surface between the locations of its end nodes. These link lengths were than used to compute the shortest path length $d_{ij}$ between all network node pairs. The resulting graph diameters are $D_g = 934$ km for Germany50 and $D_g = 6472$ km for CORONET CONUS.

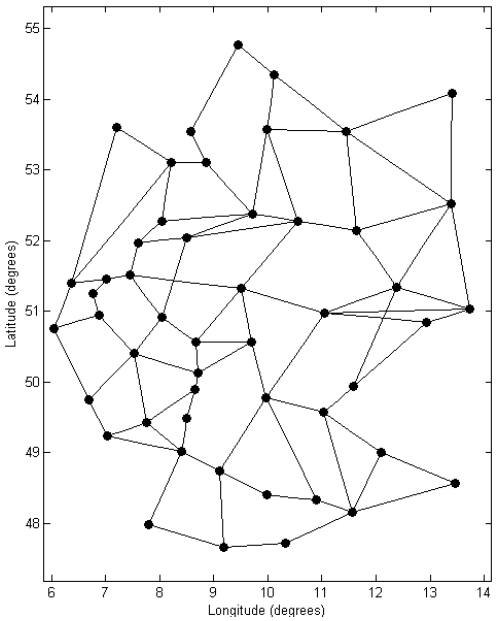


Fig. 2. Germany50 network topology with 50 nodes and 88 links

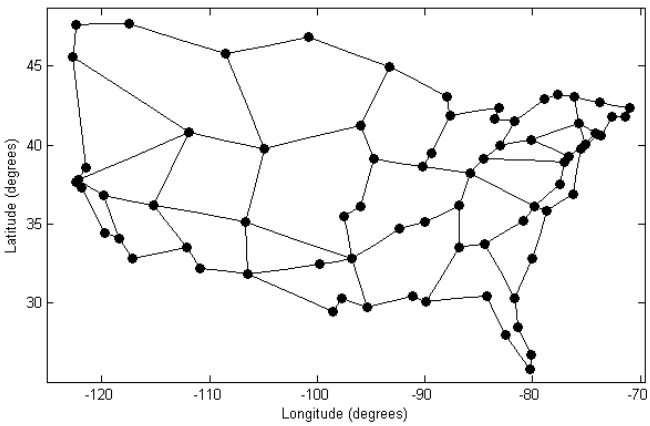


Fig. 3. CORONET CONUS network topology with 75 nodes and 99 links

In both networks, we have considered malicious attacks of $p$ = 3, 5 and 7 simultaneous nodes. Then, for each network and each value of $C = p + 1$, we have considered three sets of maximum $D_{sc}$ and $D_{cc}$ delay values representing different compromises between SC and CC delays which are tight but

still guarantee that both the non-robust and the robust CPP problems are feasible. To obtain the list of robust CPP feasible solutions, the proposed basic method plugged with the speedup technique (as described in Section III.*B*) was implemented in C++, using the CPLEX 12.6 callable library for solving the ILP models. Based on preliminary tests, $L_{\max}$ was set to 100000 and $I_{\max}$ was set to 10000. All computational results were obtained on a PC with 8 cores and 64 GB of RAM.

Table I (for Germany50) and Table II (for CORONET CONUS) present for each problem instance the number of controllers (column '*C*'), the considered maximum SC delay (column '$D_{sc}$') and maximum CC delay (column '$D_{cc}$'), the total number of feasible solutions found (column '*Nsols*'), the total runtime in seconds to find all feasible solutions (column '*Runtime (s)*'), the percentage of feasible solutions found by the speedup technique (column '*Nsols (%)*') and the percentage of the total runtime spent by the speedup technique (column '*Runtime (%)*'). Recall that when the number of feasible solutions found is $L_{\max}$, it means that the method was not able to find all existing solutions (there is only one such case, the Germany50 instance with ID=9 highlighted in Table I with an '*'). The total number of feasible solutions varies significantly between the different problem instances.

TABLE I. FEASIBLE SOLUTION RESULTS OF GERMANY50

| *ID* | *C* | $D_{sc}$ | $D_{cc}$ | *Nsols* | *Runtime (s)* | *Nsols (%)* | *Runtime (%)* |
|---|---|---|---|---|---|---|---|
| 1 | 4 | 30% | 60% | 16 | 4,9 | 87,5% | 27,9% |
| 2 | 4 | 35% | 40% | 2 | 1,2 | 50,0% | <0,1% |
| 3 | 4 | 40% | 35% | 201 | 3,5 | 98,5% | 2,4% |
| 4 | 6 | 25% | 65% | 227 | 52,5 | 93,0% | 25,9% |
| 5 | 6 | 30% | 60% | 7469 | 920,3 | 98,7% | 5,6% |
| 6 | 6 | 35% | 40% | 59 | 3,4 | 94,9% | 0,9% |
| 7 | 8 | 20% | 75% | 100 | 93,8 | 83,0% | 13,0% |
| 8 | 8 | 25% | 65% | 27603 | 7254,6 | 99,2% | 4,9% |
| 9 | 8 | 30% | 60% | 100000* | 0,8 | 99,9% | <0,1% |

TABLE II. FEASIBLE SOLUTION RESULTS OF CORONET CONUS

| *ID* | *C* | $D_{sc}$ | $D_{cc}$ | *Nsols* | *Runtime (s)* | *Nsols (%)* | *Runtime (%)* |
|---|---|---|---|---|---|---|---|
| 1 | 4 | 30% | 55% | 22 | 14,2 | 77,3% | 1,3% |
| 2 | 4 | 35% | 40% | 23 | 16,4 | 78,3% | 0,8% |
| 3 | 4 | 40% | 30% | 4 | 8,4 | 0,0% | 1,0% |
| 4 | 6 | 20% | 80% | 15 | 19,2 | 80,0% | 0,8% |
| 5 | 6 | 25% | 55% | 3 | 2,2 | 66,7% | 1,9% |
| 6 | 6 | 30% | 50% | 50 | 34,0 | 74,0% | 1,1% |
| 7 | 8 | 20% | 65% | 2109 | 1048,3 | 95,9% | 1,5% |
| 8 | 8 | 25% | 55% | 506 | 203,5 | 92,9% | 1,2% |
| 9 | 8 | 30% | 50% | 505 | 711,0 | 80,6% | 0,5% |

Concerning the merits of the speedup technique, the results of both tables show that in almost all cases, the speedup technique computes a very large percentage of solutions while using a small percentage of the total runtime. The exceptions are the Germany50 instance with ID=2 (the speedup technique found 1 out of 2 existing solutions with a negligible runtime) and the CORONET CONUS instance with ID=3 (the speedup technique did not find any of the 4 existing solutions but also took a very small fraction of the total runtime). On average, the larger the total number of solutions is, the percentage of solutions that are obtained by the speedup technique is higher while the percentage of the total runtime is lower. The extreme case is the Germany50 instance with ID=9 (100000 solutions found and all but the first solution computed by the speedup technique) where the total runtime was less than one second (mainly the runtime to solve the initial ILP model by CPLEX).

In order to compare the non-robust and the robust CPP solutions, we have also conducted the following additional computations. For the *MinAvgSC* and *MinAvgCC* non-robust CPP solutions (determined as described in Section II), we have first checked if they meet the additional robustness property (RP) that there must be a routing path from each switch to each controller that does not pass through any other controller and, then, we have computed the $n_{sc}$ and $n_s$ robustness values based on the same set of malicious node attacks as described in Section III.*C*. On the other hand, among all optimal robust CPP solutions (determined as described in Section III), we have computed the one with the minimum average SC delay and the one with the minimum average CC delay.

The obtained results for Germany50 are presented in Table III (with the solution results for the minimum average SC delays) and Table IV (with the solution results for the minimum average CC delays). Similarly, the results for CORONET CONUS are presented in Table V and Table VI. In these tables, column '*SC delay (%)*' shows the minimum average SC delay of the best solution, column '*CC delay (%)*' shows the minimum average CC delay of the best solution and column '*RP*' indicates if the robustness property is met by the non-robust CPP solution. Then, columns '$n_{sc}$' and '$n_s$' are the robustness values of the solutions for the 3 malicious node attacks as defined in Section III.*C*.

Concerning the results of Table III, they show that the average SC delays are very close between the non-robust and the robust CPP solutions. In fact, there are two instances (ID=2 and ID=3) where both solutions are equal. Concerning the robustness improvements, $n_{sc}$ improves in all other cases (*i.e.*, more switches can always connect to a surviving controller with a delay not higher than $D_{sc}$) and the improvement tends to be higher for higher values of $p$ simultaneous nodes. On the other hand, in all instances and in both the non-robust and robust CPP solutions, $n_s$ is always equal to $n - p$, which means that all surviving nodes can always connect to at least one surviving controller (this is explained by the fact that Germany50 has a high average node degree). Finally, all but one non-robust CPP solutions have the robustness property.

Concerning the results of Table IV, in this case, the non-robust and robust CPP solutions are equal in all instances for $p = 3$ (ID=1, ID=2 and ID=3). For the other cases, the conclusions follow closely the ones from the previous table. The only difference is that, although most of the CC delays are close between the non-robust and the robust CPP solutions, there is one instance where the difference is more significant: in instance with ID=6, the average CC delay increases from 22.0% to 26.1% from the non-robust to the robust CPP solution while the $n_{sc}$ improves from 39 to 42 switches (out of a total of 45 surviving switches).

TABLE III. RESULTS OF GERMANY50 FOR SC DELAYS

| | | Non-robust CPP | | | | Robust CPP | | |
|---|---|---|---|---|---|---|---|---|
| ID | p | SC delay (%) | $n_{sc}$ | $n_s$ | RP | SC delay (%) | $n_{sc}$ | $n_s$ |
| 1 | 3 | 16,0 | 44 | 47 | Yes | 16,2 | 47 | 47 |
| 2 | | 18,8 | 34 | 47 | Yes | 18,8 | 34 | 47 |
| 3 | | 17,6 | 47 | 47 | Yes | 17,6 | 47 | 47 |
| 4 | 5 | 13,4 | 28 | 45 | Yes | 14,0 | 43 | 45 |
| 5 | | 12,6 | 33 | 45 | Yes | 14,0 | 44 | 45 |
| 6 | | 17,2 | 37 | 45 | Yes | 18,2 | 42 | 45 |
| 7 | 7 | 11,0 | 29 | 43 | Yes | 11,6 | 34 | 43 |
| 8 | | 11,5 | 31 | 43 | Yes | 12,0 | 40 | 43 |
| 9 | | 11,4 | 26 | 43 | No | 12,4 | 40 | 43 |

TABLE IV. RESULTS OF GERMANY50 FOR CC DELAYS

| | | Non-robust CPP | | | | Robust CPP | | |
|---|---|---|---|---|---|---|---|---|
| ID | p | CC delay (%) | $n_{sc}$ | $n_s$ | RP | CC delay (%) | $n_{sc}$ | $n_s$ |
| 1 | 3 | 40,1 | 47 | 47 | Yes | 40,1 | 47 | 47 |
| 2 | | 29,2 | 34 | 47 | Yes | 29,2 | 34 | 47 |
| 3 | | 21,5 | 47 | 47 | Yes | 21,5 | 47 | 47 |
| 4 | 5 | 38,5 | 39 | 45 | Yes | 40,8 | 43 | 45 |
| 5 | | 29,1 | 38 | 45 | Yes | 30,8 | 44 | 45 |
| 6 | | 22,0 | 39 | 45 | Yes | 26,1 | 42 | 45 |
| 7 | 7 | 40,2 | 23 | 43 | Yes | 43,8 | 34 | 43 |
| 8 | | 31,8 | 30 | 43 | Yes | 35,9 | 40 | 43 |
| 9 | | 24,5 | 31 | 43 | No | 28,3 | 40 | 43 |

Concerning the results of Table V, recall first that these results were obtained with CORONET CONUS which has a much lower average node degree than Germany50. Once again, these results show that the average SC delays are very close between the non-robust and the robust CPP solutions. On the other hand: (i) the robustness gains are now much higher, on average, than in Germany50, (ii) there are also observable gains in the $n_s$ values and (iii) less non-robust CPP solutions have the robustness property. The most significant examples are the instances with ID=2, ID=4 and ID=7. In these instances, the robust CPP solutions have only slightly higher average SC delays (when compared with the non-robust CPP solutions) and improve the $n_{sc}$ values from 49 to 68 switches, from 49 to 61 switches and from 50 to 64 switches, respectively. These improvements represent an increase of 38.8%, 24.5% and 28.0% on the number of switches that can still work properly after any of the considered malicious node attacks.

Finally, concerning the results of Table VI, the conclusions also follow closely the ones from the previous table but the average CC delays between the non-robust and the robust CPP solutions are not as close as in the SC delay cases. Moreover, the robustness gains, although more significant than in the Germany50 case (seen in Table IV), are not as high, on average, than the ones obtained in the previous Table V.

Comparing the results of all tables, the following conclusions can be drawn. Firstly, for the same value of *p*, the robustness gains are higher for the CORONET CONUS which has a higher number of nodes. At the beginning, this might not be expected (the same *p* represents a smaller percentage of nodes being shut down in this network) but, since CORONET CONUS has a much smaller node degree, a smaller value of *p* has a higher impact on the network connectivity disruption. Therefore, the robust CPP problem is more relevant for network topologies with lower average node degrees.

Secondly, the robustness gains are small for $p = 3$ in all cases (meaning that such attacks are not too damaging on these networks) but become higher, on average, for higher values of *p*. These gains are obtained with small delay penalties when the average SC delay is considered but, in some cases, with significant delay penalties when the average CC delay is considered. The latter case, though, might not be so relevant since the maximum CC delay is the main parameter that impacts the synchronization efficiency between controllers [5] and so the average CC delay is not as important.

TABLE V. RESULTS OF CORONET CONUS FOR SC DELAYS

| | | Non-robust CPP | | | | Robust CPP | | |
|---|---|---|---|---|---|---|---|---|
| ID | p | SC delay (%) | $n_{sc}$ | $n_s$ | RP | SC delay (%) | $n_{sc}$ | $n_s$ |
| 1 | 3 | 16,4 | 69 | 71 | Yes | 16,4 | 71 | 71 |
| 2 | | 17,4 | 49 | 71 | No | 19,4 | 68 | 71 |
| 3 | | 21,4 | 56 | 71 | Yes | 21,5 | 56 | 72 |
| 4 | 5 | 11,2 | 49 | 69 | Yes | 11,3 | 61 | 69 |
| 5 | | 14,1 | 59 | 68 | Yes | 14,8 | 60 | 68 |
| 6 | | 15,2 | 65 | 68 | No | 15,5 | 65 | 69 |
| 7 | 7 | 10,8 | 50 | 52 | Yes | 11,7 | 64 | 67 |
| 8 | | 12,6 | 64 | 66 | Yes | 12,9 | 64 | 67 |
| 9 | | 14,2 | 55 | 55 | No | 14,5 | 55 | 55 |

TABLE VI. RESULTS OF CORONET CONUS FOR CC DELAYS

| | | Non-robust CPP | | | | Robust CPP | | |
|---|---|---|---|---|---|---|---|---|
| ID | p | CC delay (%) | $n_{sc}$ | $n_s$ | RP | CC delay (%) | $n_{sc}$ | $n_s$ |
| 1 | 3 | 33,1 | 67 | 71 | Yes | 42,7 | 71 | 71 |
| 2 | | 23,8 | 66 | 71 | Yes | 24,4 | 68 | 71 |
| 3 | | 19,6 | 56 | 71 | No | 22,9 | 56 | 72 |
| 4 | 5 | 46,7 | 60 | 69 | Yes | 48,1 | 61 | 69 |
| 5 | | 39,3 | 60 | 68 | Yes | 39,3 | 60 | 68 |
| 6 | | 28,0 | 32 | 34 | Yes | 30,7 | 65 | 69 |
| 7 | 7 | 36,0 | 63 | 66 | Yes | 37,2 | 64 | 67 |
| 8 | | 33,0 | 61 | 66 | Yes | 33,8 | 64 | 67 |
| 9 | | 23,8 | 32 | 34 | Yes | 26,4 | 55 | 55 |

Finally, the non-robust CPP solutions minimizing the average CC delays tend to be more robust (to malicious node attacks) than the non-robust CPP solutions minimizing the average SC delays, as shown by the higher (on average) robustness values $n_{sc}$ and $n_s$ of the *MinAvgCC* solutions when compared with the same values of the *MinAvgSC* solutions.

For illustration purposes, Fig. 4 presents a chart comparing the average SC delays of the *MinAvgSC* solutions and of the robust CPP solutions for the CORONET CONUS where it is easy to see that the delay values are very close for all instances. Fig. 5 presents the robustness gains (both in terms of $n_{sc}$ and $n_s$ values) between the same solutions. In Fig. 5, we can observe different cases: (i) large improvements in both values (ID=7), (ii) large $n_{sc}$ improvement and small (or inexistent) $n_s$ improvement (ID=2 and ID=4) and (iii) small improvements in

both values. Note that this comparison does not take into account the robustness property that is always guaranteed in the robust CPP solutions but is not observed in some *MinAvgSC* solutions (see column 'RP' in Table V). For example, the robust CPP solution of instance with ID=9 does not obtain any robustness improvement (compared with the *MinAvgSC* solution) and has a slightly worse average SC delay because the *MinAvgSC* solution does not exhibit the robustness property.

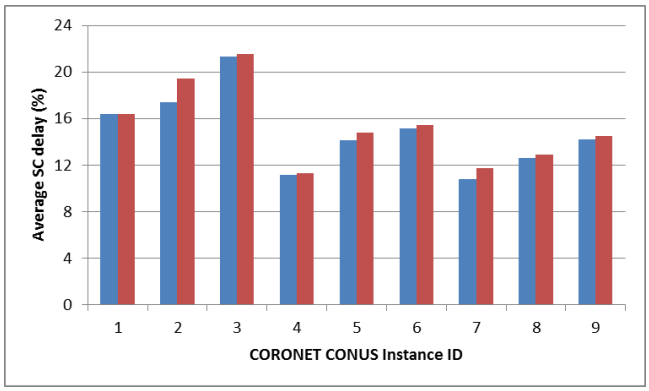


Fig. 4. Average SC delay of *MinAvgSC* solutions (in blue) and of robust CPP solutions (in red)

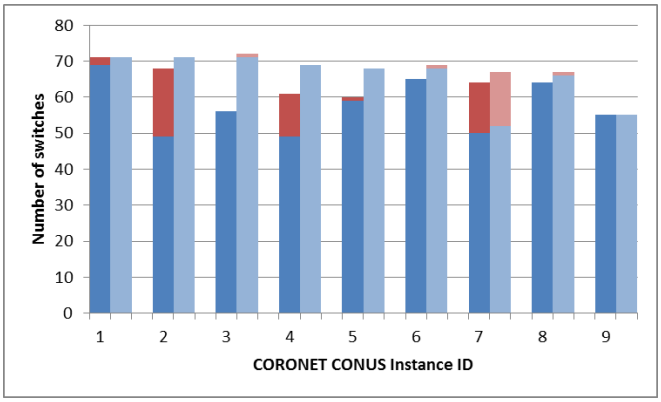


Fig. 5. Robustness values $n_{sc}$ (in dark blue) and $n_s$ (in light blue) of *MinAvgSC* solutions; robustness improvements of $n_{sc}$ (in dark red) and $n_s$ (in light red) of robust CPP solutions

## V. CONCLUSIONS

Due to the rising risk of large human-made security attacks, improving the preparedness of networks to such attacks is becoming a key issue. In this paper, we have considered the preparedness of SDN networks to multiple node malicious attacks. We have addressed a robust controller placement problem where the aim is to maximize the network robustness for a given number of malicious node attacks assuming that the attacker chooses the nodes based on centrality metrics hoping to cause maximum network disruption. We have proposed an ILP based method aiming to enumerate all feasible solutions and, then, for the different malicious node attacks, we described how to evaluate in polynomial time the solutions and determine the ones that maximize the minimum number of switches that can still be connected to at least one controller.

With a set of computational results based on two large network topologies, we have conducted a trade-off analysis between the robustness improvement of the proposed solutions against the resulting penalties on the SC and CC delays compared with CPP solutions aiming solely to minimize the average SC and CC delays. The main conclusions were that the robustness gains become more significant, on average, for sparser networks (*i.e.*, with lower average node degrees) and for attacks shutting down more nodes (although, they vary a lot between different instances). On the other hand, the average SC delay penalties are always small and the average CC delay penalties vary significantly between different instances but, in some of them, can be significant.

## ACKNOWLEDGMENT

This article is based upon work from COST Action CA15127 ("Resilient communication services protecting end-user applications from disaster-based failures – RECODIS") supported by COST (European Cooperation in Science and Technology). The work was also supported by FCT ("Fundação para a Ciência e Tecnologia"), Portugal, under the project UID/EEA/50008/2013 and through the postdoc grant SFRH/BPD/111503/2015.

## REFERENCES

[1] J. Rak, D. Hutchison, E. Calle, T. Gomes, M. Gunkel, P. Smith, J. Tapolcai, S. Verbrugge, L. Wosinska, "RECODIS: Resilient communication services protecting end-user applications from disaster-based failures", in ICTON, July 2016, We.D1.4.

[2] M. Furdek, L. Wosinska, R. Goscien, K. Manousakis, M. Aibin, K. Walkowiak, S. Ristov, J. Marzo, "An overview of security challenges in communication networks", in RNDM, Sep. 2016, pp. 43-50.

[3] B. Heller, R. Sherwood and N. McKeown, "The controller placement problem", in ACM HotSDN, New York, USA, 2012, pp. 7-12.

[4] Y. Jimnez, C. Cervell-Pastor, and A. J. Garca, "On the controller placement for designing a distributed SDN control layer," in IFIP Networking Conference, 2014

[5] D. Hock, M. Hartmann, S. Gebert, M.l Jarschel, Th. Zinner, P. Tran-Gia "Pareto-Optimal Resilient Controller Placement in SDN-based Core Networks," in ITC'13, Shanghai, China 2013.

[6] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow and G. Parulkar, "ONOS: Towards an open, distributed SDN OS," in ACM HotSDN, New York, USA, 2014.

[7] OpenDaylight: A Linux foundation collaborative project. [Online]. Available: http://www.opendaylight.org

[8] T. Zhang, A. Bianco and P. Giaccone, "The role of inter-controller traffic in SDN controllers placement", 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Palo Alto, USA, 2016, pp. 87-92.

[9] P. Vizarreta, C. Mas Machuca and W. Kellerer, "Controller placement strategies for a resilient SDN control plane", 2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM), Halmstad, Sweden, 2016, pp. 253-259.

[10] N. Perrot and T. Reynaud, "Optimal placement of controllers in a resilient SDN architecture", 2016 12th International Conference on the Design of Reliable Communication Networks (DRCN), Paris, France, 2016, pp. 145-151.

[11] D. F. Rueda, E. Calle and J. L. Marzo, "Improving the Robustness to Targeted Attacks in Software Defined Networks (SDN)", 2017 13th International Conference on Design of Reliable Communication Networks (DRCN), Munich, Germany, 2017, pp. 1-8.

[12] G. Nencioni, B. E. Helvik and P. E. Heegaard, "Including Failure Correlation in Availability Modelling of a Software-Defined Backbone Network", to appear in IEEE Transactions on Network and Service Management.