Survivor: an Enhanced Controller Placement Strategy for Improving SDN Survivability

Lucas F Müller, Rodrigo R Oliveira, Marcelo C Luizelli,
Luciano P Gaspary, Marinho P Barcellos
Federal University of Rio Grande do Sul, Institute of Informatics, Porto Alegre, RS, Brazil
Email: {lfmuller, ruas.oliveira, mcluizelli, paschoal, marinho}@inf.ufrgs.br

***Abstract*—In SDN, forwarding devices can only operate correctly while connected to a logically centralized controller. To avoid single-point-of-failure, controller architectures are usually implemented as distributed systems. In this context, recent literature identified fundamental issues, such as device isolation and controller overload, and proposed controller placement strategies to tackle them. However, current proposals have crucial limitations: (i) device-controller connectivity is modeled using single paths, yet in practice multiple concurrent connections may occur; (ii) peaks in the arrival of new flows are only handled on-demand, assuming that the network itself can sustain high request rates; and (iii) failover mechanisms require predefined information, which, in turn, has been overlooked. This paper proposes Survivor, a controller placement strategy that addresses these challenges. The strategy explicitly considers path diversity, capacity, and failover mechanisms at network design. Comparisons to the state-of-the-art on survivable controller placement show that Survivor is superior because (a) path diversity increases the survivability significantly; and (b) capacity-awareness is essential to handle overload during both normal and failover states.**

I. Introduction

Software-Defined Networking defines a new architecture in which the control plane of network forwarding devices is moved to a logically centralized controller. Although beneficial for network management, SDN created an inherent dependency relationship between the forwarding devices and the controller. More precisely, devices need to remain connected to the controller in order to operate properly, otherwise they apply outdated policies and fail to deal with unspecified flows [1]. Therefore, a key challenge is to ensure the connectivity between controller and devices in face of harmful events in the network. Hereafter, this challenge will be referred to as *control plane survivability* (or *survivability*, for short).

SDN architectures avoid single-point-of-failure by implementing controllers as distributed systems [2], [3]. Although replication increases survivability, fundamental aspects make it insufficient. Particularly, two main issues must be properly addressed. First, disruptions in the network can physically isolate forwarding devices from controller instances. Second, high network demand may overload a controller replica, negatively affecting responsiveness.

In order to tackle these issues, literature has proposed different controller placement strategies [4]–[6]. However, these proposals have three main limitations. First, connections between forwarding devices and controller instances are modeled using single paths; yet, in practice multiple concurrent connections may occur. Second, changes in traffic load (such as churn in the arrival of new flows) can only be handled on-demand; this aspect assumes that the network has been planned in advance for high request rates. Lastly, failover mechanisms depend on predefined lists of backup controllers; such a list is currently composed ad-hoc, which, in turn causes these mechanisms to behave inefficiently.

This paper proposes *Survivor*, a novel controller placement strategy that overcomes the above shortcomings. Survivor has three main benefits. First, connectivity is enhanced by explicitly considering path diversity. Second, controller overload is avoided proactively by adding capacity-awareness in the controller placement. Third, failover mechanisms are improved by means of a methodology for composing their list of backups. Comparisons are performed to evaluate the performance of the proposed solution. The main contributions of this paper are threefold:

- **Significant reduction on connectivity loss.** Correctly exploring the path diversity of the network (i.e., connectivity-awareness) reduces the probability of connectivity loss in *around 66%* for single link failures. Moreover, simply adding path diversity to current solutions is not sufficient; the proposed solution still presents *between 2 and 3 times less* disconnected devices in worst case failures.
- **More realistic controller placement strategy.** Survivor is the first strategy to consider capacity-awareness proactively, since previous work handled requests churn on-demand. Experiments show that capacity planning is essential to avoid controller overload, specially during failover. Additionally, the placement strategy is implemented as an optimization model in order to generate *optimal results*.
- **Smarter recovery mechanisms.** Survivor encompasses heuristics for defining a list of backup controllers. A methodology for composing such lists, modeled as a generic heuristic framework, is developed and two heuristics are implemented. As a result, the converging state of the network can improve considerably depending on the selected heuristic.

The remainder of this paper is organized as follows: Section II introduces the fundamental concepts related to this

paper and discusses the main limitations of related work. Section III presents the proposed strategy, while experiments are evaluated in Section IV. Section V discusses the final considerations and indicates possible future work.

II. Background and Related Work

This section begins by identifying the considered set of failures, and currently available failover mechanisms.

A. Failures and Failover Mechanisms

SDN establishes a dependency relationship between the controller and the forwarding devices. More specifically, the lack of connectivity between both systems causes network malfunction, because devices get outdated and cease to receive instructions on how to forward new flows. Therefore, it is important to analyze possible failure scenarios and the available failover mechanisms[1].

Failure scenarios. In line with previous work [4], [5], the failures considered in this paper occur independently and in both the data plane and control plane. In the data plane, failures arise upon disruptions in links or forwarding devices. In turn, control plane failures happen due to software or hardware malfunction on the machine hosting the controller. Pragmatically, in both cases the result of a failure is the loss of connectivity between a set of routing devices and controller instances.

Failover mechanisms. The OpenFlow Protocol [8] provides mechanisms that allow a forwarding device to react upon loss of connectivity with the controller. Towards this end, devices maintain *auxiliary connections* and a *list of backup controllers*. Auxiliary connections with the controller can be defined, established and kept open in order for the device to avoid unnecessary delays. Moreover, they can be established over any path in the network that reaches the controller (e.g., link/node-disjoint paths, shortest-paths, widest-paths). In turn, the list of backup controllers defines the order in which the device will attempt to connect to different controller instances. This mechanism is used when all connections to the primary controller instance are lost (either due to massive link failures or controller failure).

B. Survivable Controller Placement

The Controller Placement Problem [9] is an optimization problem which consists of finding the best placement for controller instances such that a given metric is optimized. Accordingly, the *Survivable* Controller Placement consists of finding the best placement that optimizes a survivability-related metric. This subsection reviews work that attempt to maximize control plane survivability according to two metrics: network disruptions [4], [5] and controller overload [6].

In respect to disruptions, current controller placements minimize the likelihood of disconnection. Zhang et al. [4] defines a min-cut algorithm that specifies clusters of forwarding devices and then places one controller instance in each cluster's centroid. In turn, Hu et al. [5] chooses controller instances such that the chance of connectivity loss is minimized. In both strategies, connections are defined according to the shortest-path between controller instances and forwarding devices.

In contrast, Bari et al. [6] study the dynamic placement of controller instances over the network topology. The authors consider that the initial position is somehow predefined and are limited to turn on/off controllers according to a particular network load. Hence, controller-device connections are defined dynamically and on-demand.

C. Limitations with Current Proposals

Current failover mechanisms (§II-A) allow devices to recover from some link/node/controller failures. When a failure breaks the primary connection to a controller, the forwarding device may use pre-calculated auxiliary connections. If the disruption renders the controller instance unavailable, a device follows a predefined list in order to attempt to connect to other controller instances. The mechanics on how to define auxiliary connections and the list of backup controllers are currently unspecified.

In regard to controller placement strategies (§II-B), literature proposes to tackle either controller-device disconnections or controller overload. While the former is currently limited to using shortest-paths only, the latter assumes that controllers have been already optimally placed and simply determines the assignment of devices to controllers. In sum, a proper solution needs to efficiently recover from controller-device disconnections whenever feasible and optimally position controllers to reduce the chance of overload before and after failures.

III. Survivor

A. Overview

As previously discussed, Survivor deals with three main aspects: connectivity, capacity, and recovery. Towards this end, the strategy is divided in two parts. The first defines placement for controller instances, while the second specifies a list of backup controllers for each device in the network. Both parts are expressed as optimization problems.

Placing controller instances. The first part requires finding optimal placements for controller instances such that *connectivity is maximized* and *capacity constraints are satisfied*. In order to maximize connectivity, the algorithm chooses positions that yield the highest number of node-disjoint paths between the forwarding devices and the controller instance they connect to. In turn, dealing with resource constrains requires some intuition about device demands and controller capacity. Estimates for device demands can be obtained with network measurements [10], while controllers capacity can be inferred through experimentation [11]. In addition, upon disruptions in the network, some forwarding devices may be moved from one controller instance to another. This situation may cause a controller to get overloaded, leading to request

[1]This paper only considers failover mechanisms that help reestablishing the controller-device connections. For mechanisms that provide minimal functionality when connections cannot be reestablished, the interested reader may refer to the data plane connectivity literature [7] or the OpenFlow Protocol Specification [8].

delays or losses. To sustain exceeding demands, each controller has a percentage of capacity reserved as backup. This algorithm is implemented using Integer Linear Programming (ILP), which guarantees optimality. The model is presented in Section III-B.

Composing a list of backup controllers. The second part consists in defining an ordered list for each device using a given heuristic. Interestingly, several heuristics can be composed using the same principle: optimize a local metric and generate an ordered list. Accordingly, the heuristics used in this paper were implemented using a generic framework. This framework, its related assumptions, and the implemented heuristics are described later (§III-C).

B. Implementation of an Optimal Controller Placement

Input. Tuple $I = \{G(N, L);\ C;\ U_i;\ R_i;\ \mathcal{DP}_{i,j};\ \alpha_i\}$ represents the input of the ILP. The physical topology is denoted by an undirected graph $G = (N, L)$, where nodes $n \in N$ represent forwarding devices and edges $(n, m) \in L$ represent bidirectional links. The set of possible controller instances is given by C. The capacity of each controller $c \in C$ is denoted by U_c, while the request demand of each device n is represented by R_n. Additionally, $\mathcal{DP}_{n,m}$ gives the pre-calculated number of node-disjoint paths between nodes n and m. Finally, $\alpha_c : c \in C$ indicates the percentage of backup capacity set to each controller.

Output. The tuple $V = \{x_{i,j};\ y_{i,j};\ w_{i,j}\}$ represents the variables of the ILP. Device mappings are given by $x_{n,c} \in \{0, 1\}$; they indicate whether device n is mapped to controller c. Controller placements are denoted by $y_{c,n} \in \{0, 1\}$; they indicate whether controller c is placed on top of (i.e., is physically connected to) node n. Lastly, $w_{c,n} \in \mathbb{N}$ counts the number of disjoint paths between controller c and device n; this last variable holds 0 when n is not mapped to c.

Objective. The general goal of the proposed strategy is to maximize connectivity between forwarding devices and controllers instances. This goal is modeled as equation

$$\max \quad \frac{\sum_{c \in C} \sum_{n \in N} w_{c,n}}{|N|}, \tag{1}$$

which maximizes the average of disjoint paths between devices and their controller.

Constraints. The constraints of this ILP model can be divided into three categories: *placement-related*, *capacity-related*, and *connectivity-related*.

The first four constraints [Constr. (C1)–(C4)] are *placement-related*. They ensure correctness for both the placement of controller instances in the topology and the mapping of devices to controller instances. Constraint (C1) guarantees that, for all devices (i.e., $\forall n \in N$), each device n will be controlled by exactly one controller c (i.e., $\sum_{c \in C} x_{n,c} = 1$).

$$\sum_{c \in C} x_{n,c} = 1 \qquad \forall n \in N. \ \text{(C1)}$$

TABLE I
SUMMARY OF SYMBOLS FOR THE ILP MODEL.

Symbol	Definition
N	Set of network nodes (i.e., forwarding devices).
L	Set of links.
C	Set of controllers.
U_c	Maximum number of requests that controller c can handle.
R_n	Number of requests of each device n.
$\mathcal{DP}_{n,m}$	Number of disjoint paths between node pairs n and m.
α_c	Percentage of capacity reserved as backup in controller c.
$x_{n,c} \in \{0, 1\}$	Indicates whether device n is mapped to controller c.
$y_{c,n} \in \{0, 1\}$	Indicates whether controller c is placed onto node n.
$w_{c,n} \in \mathbb{N}$	Indicates the number of disjoint paths between device n and controller c. *(0 if device n is not mapped to controller c)

Constraint (C2) is twofold: (i) assigned controllers must be active (i.e., $x_{n,c} \leq \sum_{m \in N} y_{c,m}$); and (ii) a controller cannot be placed in more than one location (i.e., $\sum_{m \in N} y_{c,m} \leq 1$).

$$x_{n,c} \leq \sum_{m \in N} y_{c,m} \leq 1 \qquad \forall c \in C, \forall n \in N. \ \text{(C2)}$$

Constraint (C3) ensures that two controllers will not be placed in the same location, while Constraint (C4) guarantees that if the controller c is placed onto node n (i.e., $y_{c,n} = 1$), then the device n should be mapped to controller c (i.e., $x_{n,c} = 1$).

$$\sum_{c \in C} y_{c,n} \leq 1 \qquad \forall n \in N; \ \text{(C3)}$$

$$y_{c,n} \leq x_{n,c} \qquad \forall n \in N, \forall c \in C. \ \text{(C4)}$$

The *capacity-related* constraint [Constr. (C5)] ensures that the controller capacity will not be exceeded. It considers both normal and backup capacities in the same formulation. Specifically, for all controller instances (i.e., $\forall c \in C$), a controller normal capacity [i.e., $(1 - \alpha_c) \cdot U_c$] will not be exceeded by the sum of all demand assigned to it (i.e., $\sum_{n \in N} x_{n,c} R_n$).

$$\sum_{n \in N} x_{n,c} R_n \leq (1 - \alpha_c) \cdot U_c \qquad \forall c \in C. \ \text{(C5)}$$

The two final constraints [Constr. (C6.1) and (C6.2)] are *connectivity-related*. They are used to count the number of disjoint paths in each controller-device mapping. These constraints are trickier to interpret, as they work in conjunction and depend on the objective function. The constraints represent two cases. When device n is mapped to controller c ($x_{n,c} = 1$), $w_{c,n}$ is upper bounded by the disjoint paths between m and c's placement (n), which is given by $\mathcal{DP}_{n,m}$ [Constr. (C6.1)]. In contrast, when device n is not mapped to controller c ($x_{n,c} = 0$), $w_{c,n}$ is upper bounded by 0 [Constr. (C6.2)]. In short, Constraint (C6.1) is always set to $\mathcal{DP}_{n,m}$, while Constraint (C6.2) is either 0 or infinity, depending on $x_{n,c}$.

$$w_{c,n} \leq \sum_{m \in N} y_{c,m} \cdot \mathcal{DP}_{n,m} \qquad \forall c \in C, \forall n \in N; \ \text{(C6.1)}$$

$$w_{c,n} \leq x_{n,c} \cdot \kappa \qquad \forall c \in C, \forall n \in N : \kappa \to \infty. \ \text{(C6.2)}$$

Constraints (C6.1) and (C6.2) set upper bounds for $w_{c,n}$, which are the exact values that it should hold. In other words,

$w_{c,n}$ should always be assigned its upper bound. This is ensured by the objective function [Eq. (1)], which always tries to maximize the value for $w_{c,n}$.

C. Heuristics for Defining Lists of Backup Controllers

This subsection first introduces the assumptions and notations used as base for the algorithms. Next, it describes a generic framework for designing heuristics that compose the lists of backup controllers, eliminating the need to manually determine the list. Then, it presents two heuristics as proof of concept, one based on proximity and the other based on residual capacity.

Assumptions. The generic framework makes three assumptions. First, controller failures are considered the worst cases for disconnections; while this is not always true, it can be expected for all practical purposes since, as demonstrated in experiments (§IV), the proposed strategy greatly reduces the chance of disconnections due to other disruptions. Second, controllers are assumed to fail independently; this can also be presumed in practice, as current architectures implement mechanisms to ignore controller instances when they fail, protecting the remaining ones [2], [3]. Third, each index on the list is considered to be independent, as all instances will follow their ordered list; in other words, all devices are expected to use the same index (mostly the first) on the same failure scenario.

Notation. Most notations used in the following algorithms are intuitive. However, some considerations must be made. Particularly, *brackets* ([.]), *star* ($*$), and *plus sign* ($+$) have special meanings. *Brackets* are used to define indices in ordered lists, that is, $[i]$ indicates an earlier position than $[j]$ for all $i < j$. *Star* is used to indicate that a given operation is performed to all indices of a list. Lastly, *plus sign* is used to indicate that a comparison is performed to all indices of a list, subject to at least one comparison being true.

Generic framework. The pseudo-algorithm of the framework is described in Algorithm 1. It begins by considering all controller instances that were placed in the topology (line 3). Due to the first and second assumptions, controllers are evaluated independently. Next, for each index on the list (line 4), it initializes a list of candidates, which is composed by all controllers except the one being evaluated (line 5). Similarly to controllers, indices are evaluated independently due to the third assumption.

The following steps select all devices connected to the evaluated controller and optimize the i^{th} index on their list (lines 6-10). Three operations are performed: *select local optimum* (line 8), *set list index* (line 9), and *update candidate list* (line 10). The *select local optimum* operation consists of evaluating all possible candidates, given by set S (i.e., $S \leftarrow K \setminus B_n[*]$), and then selecting the best among them according to a given metric (i.e., $\varphi(n, S) = b$).

Set list index is solely an attribution operation. In contrast, the *update candidate list* operation follows a specification (i.e., δ) to update information about the controllers of the candidate list. It is important to realize that operations performed by the update procedure are temporary, as they are performed in set K, which, in turn, is reset in line 5. The aforementioned operations use two generic procedures: $\varphi(n, S)$ (which finds the local minimum) and δ (which updates candidates). These procedures have specific meanings for each heuristic and will be exemplified in the following.

Algorithm 1: Generic heuristics framework for composing the lists of backup controllers

```
1: B_n[*] ← ∅, ∀n ∈ N
2: ListLength ← max(|B_*|)
3: for c ∈ C : y_{c,+} = 1 do // only get placed controllers
4:    for i ← 1 upto ListLength do
5:       K ← C \ {c} // initialize the list of candidates
6:       for n ∈ N : x_{n,c} = 1 do // only get connected devices
7:          S ← K \ B_n[*] // discard already used
8:          select b ∈ S : φ(n, S) = b // get local optimal
9:          B_n[i] ← b // set b in the i^th index of n's list
10:         update K according to rule δ
11:      end for
12:   end for
13: end for
14: return B_n
```

Proximity-based heuristic. This heuristic attempts to select the closest controller instances to use as backup (in terms of delay or hops). To implement this heuristic, one is required to define $\varphi(n, S)$ as a procedure that takes device n and the set of valid controller candidates S, and returns the closest instance. This can be achieved by implementing a shortest-paths algorithm inside $\varphi(n, S)$. The update rule is not required since controllers do not change positions.

Residual capacity-based heuristic. This heuristic attempts to account for resource consumption while selecting controller instances. More specifically, it *selects* the controller instance that has the highest residual capacity, and *updates* this instance according to the exceeding demand. Accordingly, $\varphi(n, S)$ is a linear search for the maximum residual capacity, while δ adds the demand of device n to controller n in set k. As previously stated, operations performed by the update procedure are temporary.

IV. Evaluation

A. Methodology

This section introduces the methodology employed to compare Survivor with the state-of-the-art on survivable controller placement.

Workload. The analysis takes as input the network design given by each placement strategy and then simulates multiple failure scenarios. Three different WAN topologies were considered[2]: Internet2 (10 nodes, 15 links), RNP (27 nodes, 33 links) and GÉANT (40 nodes, 61 links). These environment assumes that nodes would be OpenFlow-capable devices and that controllers could be provisioned at any of these nodes' locations. Additionally, capacities and demands were based on studies from literature. All controllers have the same capacity,

[2]Topology maps were acquired in January 2014, from the Internet Topology Zoo: http://www.topology-zoo.org/

1800 kilorequests/s [11], while each forwarding device in turn generates 200 kilorequests/s [10]. Finally, the percentage of resources set as backup is 30%.

Comparison Method. Survivor (SVVR) is compared with two versions of the one developed by Zhang et al. [4]. The original version of the algorithm is composed by two steps: (i) identify partitions in the network topology with minimum cuts across boundaries; and (ii) assign one controller to the location which has the shortest paths to all devices in the same cluster. This strategy is denoted *MCC*, which stands for MinCut-Centroid. Additionally, a natural extension of this algorithm, denoted *MCC+*, consists of considering all available paths between controller and devices. The algorithm runs as usual, but after controller instances are placed, connections are made using all available node-disjoint paths.

Metrics. Resulting topology designs are analyzed in terms of resilience to disruptions and overload. Resilience represents the capacity of the evaluated strategies to sustain loss of connectivity upon controller or link disruptions. In its turn, overload indicates how controller instance loads are distributed during normal operation and after failures. Four metrics are used, the first two quantify resilience, while the last two measure overload. For fairness of comparison, the first evaluation considers the same *resilience equation* used by Zhang et al. [4]; this equation measures the probability of connectivity loss in a uniform failure scenario. The second metric uses *cardinal of edge-connectivity* [12] to calculate the percentage of disconnected components given all possible failure scenarios. The third metric counts the *number of overloaded controllers* on all possible failure scenarios. Finally, the fourth metric shows the *load distribution* for each of the controller instances.

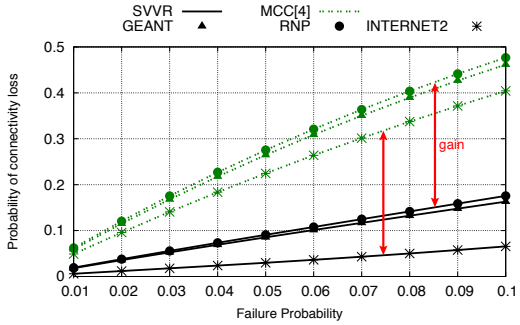
B. Results

This section compares the Survivor (SVVR) against literature (MCC/MCC+) using the four metrics presented earlier. For ease of exposure, Survivor is illustrated using full-black lines, whereas MCC (or MCC+) using dotted lines (which are green, if color is available).

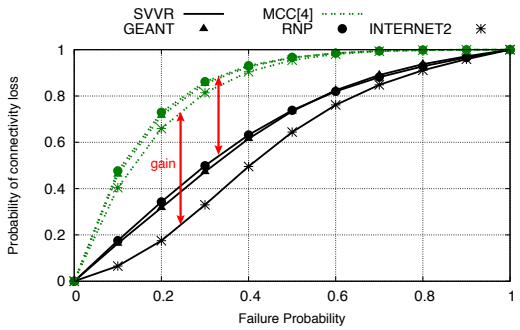
1) Survivor reduces the probability of connectivity loss: The first evaluation (shown in Fig. 1) considers that network elements (i.e., nodes and links) fail independently and then measures the probability of a controller-device connection be interrupted. Axis x represent the failure probability assigned to all network elements (that is, 0.01 means all nodes and links have 1% chance of failing); in turn, the y axis yields the probability of a connectivity loss. Fig. 1(a) shows values from 1% to 10% (the same used by Zhang et al. [4]), while Fig. 1(b) extrapolates the analysis to the entire range (0% to 100% probability of failure).

Fig. 1(a) shows that Survivor outperforms MCC consistently, and also substantially, for all probabilities (in the x axis) but the extremes 0 and 1. More importantly, the curves representing Survivor behavior have a slower growing factor than MCC. As can be observed, when elements have 1% chance of failure both strategies behave similarly (less then 8% chance of connectivity loss). As the chance of failure increases to 5%, the probability of connectivity loss for Survivor is less than half of that for MCC. In this case, Survivor has stayed bellow 10%, while MCC has increased above 20%. Moreover, although the relative difference between both strategies decrease, Fig. 1(b) shows that Survivor reaches near 100% of connectivity loss much later than MCC. While the probability of connectivity loss of Survivor is still around 80% when the chance of failure is 60%, MCC is already near 100%.

The observed improvement happens because in MCC a single element failure can break one or multiple controller-device connections; yet in Survivor, it is required at least one element failure per controller-device path. This seems to be a direct effect of exploring path diversity during placement. The next experiment provides further evidence of this behavior and shows that considering path diversity after placement is insufficient.



(a) Probability distribution 0.01% to 10%.



(b) Probability distribution 0% to 100%.

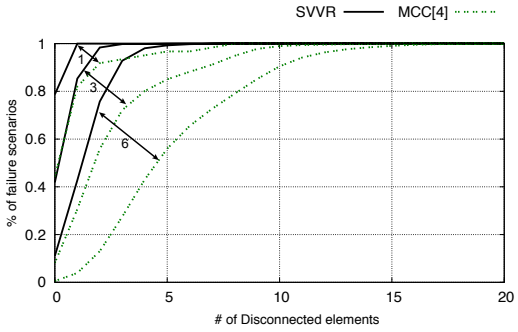
Fig. 1. Probability of controller-device connectivity loss for different failure probabilities. Values are obtained using the equation in [4]. *The lower the curve, the better.*

2) Path diversity increases the network survivability, and it requires explicit consideration to be fully explored: Fig. 2 expresses a CDF of the number of disconnected elements, when enumerating all possible failure scenarios. Three variations are evaluated: single link failure, 3 concurrent link failures, and 6 concurrent link failures. These variations will be denoted $k = 1$, $k = 3$, and $k = 6$, respectively. Moreover, results were similar for all topologies, thus, due to space constraints, only the larger one – GÉANT – is shown.

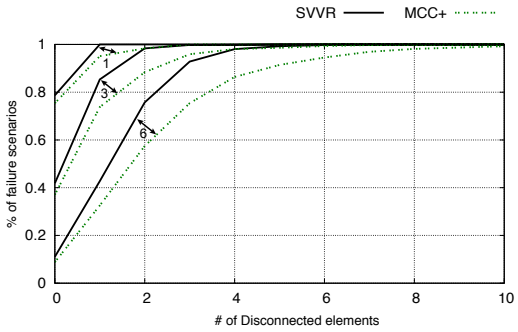
Results in Fig. 2(a) compare Survivor to MCC. For single link failures, $\text{SVVR}_{k=1}$ achieves twice the protection (80%)

than $\text{MCC}_{k=1}$. Moreover, the worst case in $\text{SVVR}_{k=1}$ is only one disconnected device, while in $\text{MCC}_{k=1}$ there may be up to 8 disconnected devices. *Finding: the probability of a single link failure affecting all connections of even one device is much smaller (*$100\% - \frac{\sim 20\%}{\sim 60\%} \approx 66\%$ *less) than that of affecting a single connection of one or multiple devices.*

Fig. 2(a) also shows that Survivor in the scenario with 3 and 6 concurrent link failures occasionally outperforms MCC in single link failures. $\text{SVVR}_{k=3}$ and $\text{MCC}_{k=1}$ behave similarly in 80% of cases, but $\text{SVVR}_{k=3}$ outperforms $\text{MCC}_{k=1}$ in the remaining 20%. More importantly, the remaining 20% represent worst case scenarios; in such scenarios $\text{SVVR}_{k=3}$ disconnects at most 6 devices, whereas $\text{MCC}_{k=1}$ disconnects up to 8. Additionally, for $\text{SVVR}_{k=6}$ and $\text{MCC}_{k=1}$, there is a turning point around 90%. When considering 6 concurrent failures for both, this difference is more noticeable. In $\text{SVVR}_{k=6}$, the worst 20% disconnection cases have between 3 and 11 disconnected devices; in $\text{MCC}_{k=6}$, the same $[3, 11]$ interval happens in 80% of the cases (from 10% up to 90%). In this interval, Survivor is 75% less likely to disconnect the same amount of devices ($100\% - \frac{\sim 20\%}{\sim 80\%} \approx 75\%$).



(a) Survivor x MCC.



(b) Survivor x MCC+.

Fig. 2. Cumulative distribution function of disconnected devices for all possible cases of 1, 3, and 6 link disruptions in the GÉANT topology. *Curves closer to the top-left side are better.*

Finally, in order to evaluate the effectiveness of the Survivor placement strategy, path diversity is added to MCC after placement; this modification is denoted MCC+. Fig. 2(b) shows that Survivor still outperforms MCC+ in all cases. *Finding: adding path diversity after placement in insufficient, as MCC+ has 2-3 times more disconnected devices in worst cases*; $\frac{3}{1} = 3$, $\frac{19}{6} \approx 3$, and $\frac{22}{11} = 2$, for $k = 1$, $k = 3$, and $k = 6$, respectively ($\frac{worstMCC+}{worstSurvivor}$).

3) Network convergence after disruptions is highly sensible to predefined information in failover mechanisms: Fig. 3 shows the number of overload scenarios considering all controller instances. Each overload scenario counts as a single controller instance that had a load higher than 100%. The analysis considers three cases of network operation: normal, post-failure with proximity-based failover heuristic, and post-failure with residual capacity-based failover heuristic. In normal operation Survivor has not a single overloaded controller instance. This happens because the placement strategy is capacity-aware and, thus, avoids overloads when assigning devices to controllers. In contrast, MCC has demonstrated a small number of overloaded controller instances during normal operation. This behavior arises from the fact that MCC only considers a centroid heuristic when assigning devices to controllers. Hence, in some cases, a controller instance happens to have a much higher load.

After disruptions, the failover mechanism employed the lists created by the two heuristics proposed earlier (§III-C). Evaluation shows that failover mechanisms perform very differently depending on the way their backup lists were composed. The residual capacity-based heuristic outperformed the proximity-based one for both MCC and Survivor. Even though Survivor had set backup resources, some instances were still overload in the proximity-based heuristic. This behavior becomes more evident in the next evaluation.

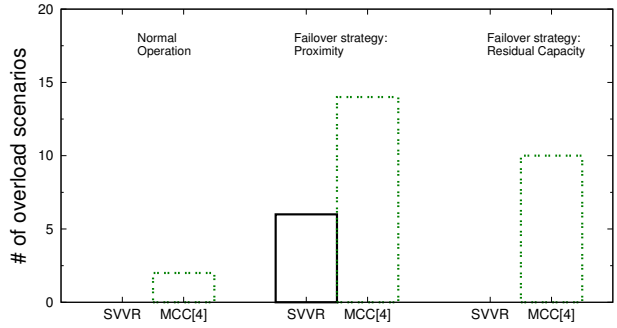


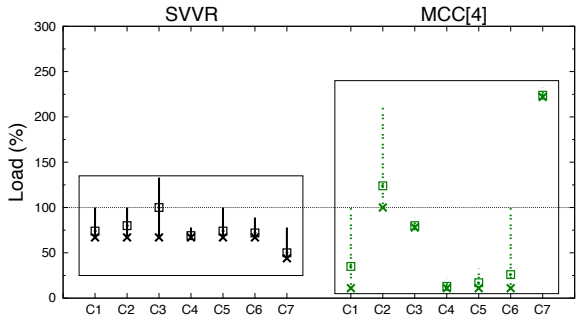
Fig. 3. Number of overload scenarios (controller load $>100\%$) during network normal operation and two post-failover operation cases. *The lower the bars, the better.*

4) Controller overload can be handled proactively by adding capacity-awareness and setting backup resources: The previous evaluation shown that different kinds of information used during failover yield sensible performance impact. This evaluation takes a closer look in the state of the network after convergence to better understand the observed behavior. Towards this end, Fig. 4(a) shows the load on each controller considering all failure scenarios. Values indicate the minimum, maximum, average and mode loads.

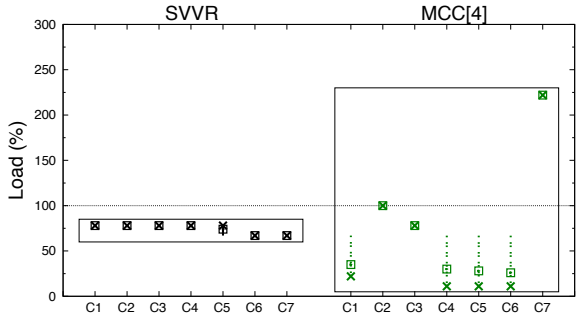
As observed, even though placements end up with enough free capacity, the proximity heuristic (Fig. 4(a)) tends to overload some instance(s) (e.g., C3 in Survivor and C1, C2, and C6 in MCC). For the Survivor placement, the maximum and minimum values have high differences for all controllers

and the average varies, but the mode in particular shows that controller instances have their load close to the minimum in most cases. This happens because most devices select the same backup controller when their primary controller fails, thus leaving other controllers with their original load. The MCC placement shows similar results, but the variation among values tends to be higher because the initial placement is already unbalanced.

In contrast, the residual capacity heuristic (Fig. 4(b)) tends to leave the load more evenly distributed. In the Survivor placement, the maximum, minimum, average, and mode have almost the same value. This happens because devices will attempt to connect to different controllers in the network, and will prefer those with higher residual capacity. As a result, the load on controllers increases, but none of them overloads. In this case, MCC behaves differently than Survivor. This can be explained by the fact that in MCC, controllers C2, C3, and C7 were assigned a higher load (in comparison to the other controllers) during the initial placement – in fact, C2 and C7 were overloaded. As a result, controllers C1, C4, C5, and C6 were the only ones assigned during failover and their load variations are similar.



(a) Proximity-based heuristic for failover mechanism.



(b) Residual Capacity-based heuristic for failover mechanism.

Fig. 4. Mininum, maximum, average (square), and mode (star) values for the load in each controller instance (labeled C1-C7) after failover. Placement strategies (Survivor and MCC) are boxed for ease of exposition and comparison. Results are shown for the GÉANT topology.

V. Conclusion and Future Work

SDN has altered the requirements for network survivability, as the main challenge became maintaining the connectivity between the controller and the forwarding devices upon disruptions in the network. Previous work has relied on strong assumptions in order to simplify the problem. This paper presented Survivor, a novel approach that is shown to improve survivability in SDN. The main improvements in comparison to previous work are: (a) considering multiple paths explicitly; (b) dealing with capacity during initial placement; and (c) developing smarter failover mechanisms.

Main findings showed that: (i) exploring path diversity during placement reduces the chance of connectivity loss; (ii) adding path diversity to current solutions was not sufficient; (iii) adding capacity-awareness to the solution was essential, otherwise controller instances might get overloaded on both normal and failover scenarios; and (iv) the heuristic for selecting backups during failover has substantial impact on the converging state of the network. In future work, the evaluation should be extended to consider specific environments, such as other topologies and failure scenarios. Additionally, more aspects could be explored in order to further improve survivability.

Acknowledgment

This work has been supported by FP7/CNPq (Project SecFuNet, FP7-ICT-2011-EU-Brazil).

References

[1] S. Vissicchio, L. Vanbever, and O. Bonaventure, "Opportunities and research challenges of hybrid software defined networks," *ACM Computer Comm. Review*, vol. 44, no. 2, April 2014.

[2] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: a distributed control platform for large-scale production networks," in *Proc. OSDI*. USENIX, 2010, pp. 1–6.

[3] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proc. SIGCOMM HotSDN workshop*. ACM, 2012, pp. 19–24.

[4] Y. Zhang, N. Beheshti, and M. Tatipamula, "On resilience of split-architecture networks," in *Proc. GLOBECOM*. IEEE, 2011, pp. 1–6.

[5] Y. Hu, W. Wendong, X. Gong, X. Que, and C. Shiduan, "Reliability-aware controller placement for software-defined networks," in *Proc. IM*. IEEE, 2013, pp. 672–675.

[6] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks," in *Proc. CNSM*. IEEE, 2013, pp. 1–8.

[7] J. Liu, A. Panda, A. Singla, B. Godfrey, M. Schapira, and S. Shenker, "Ensuring connectivity via data plane mechanisms," in *Proc. NSDI*. USENIX, 2013, pp. 113–126.

[8] ONF, "Openflow switch specification 1.4.0," Available at <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf>, 2014.

[9] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proc. SIGCOMM HotSDN Workshop*. ACM, 2012, pp. 7–12.

[10] I. Cunha, F. Silveira, R. Oliveira, R. Teixeira, and C. Diot, "Uncovering artifacts of flow measurement tools," in *Proc. PAM*. Springer-Verlag, 2009, pp. 187–196.

[11] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proc. Hot-ICE Workshop*. USENIX, 2012, pp. 10–10.

[12] K. Bagga, L. Beineke, R. Pippert, and M. Lipman, "A classification scheme for vulnerability and reliability parameters of graphs," *Mathematical and Computer Modelling*, vol. 17, no. 11, pp. 13 – 16, 1993.