Fault Tolerant Controller Placement in Distributed SDN Environments

Adel Alshamrani, Sayantan Guha, Sandeep Pisharody, Ankur Chowdhary, Dijiang Huang
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University
Tempe, AZ 85281
Email: aalsham4, sguha3, spishar1, achaud16, dhuang8@asu.edu

***Abstract*—Software Defined Network (SDN) facilitates a centralized networking system where a controller manages the global view of the network. The introduction of Software-Defined Networks and standards such as OpenFlow spawn several questions regarding scalability and reliability. One such question is the controller placement problem; i.e. given a topology, the problem of determining how many controllers are needed, and where they should be placed. This question has been well-studied relative to performance, but there has not been a focus on maximizing fault-tolerance. In this paper, we present a model for controller placement to account for fault-tolerance and compare our algorithm to existing algorithms. Our proposed solution was analyzed to determine where controllers should be placed on a wide range of topologies from the Internet Topology Zoo. We further evaluated the dependence of fault-tolerance over the range of available number of controllers.**

***Keywords*—Software-Defined Network (SDN), OpenFlow, Controller, Controller placement, Fault Tolerance (FT), Metric**

I. Introduction

Software-Defined Networks (SDNs) are a recently emerging networking paradigm whose primary advantage is giving developers a larger amount of control over the way network traffic is handled and to change the limitation of current network infrastructures. The notable distinction between an SDN and a traditional network is that the control logic is separated from underlying routers and switches that forward the traffic. This separation between the definition of the network policies, their implementation in switching hardware, and the forwarding traffic is a notable distinction that leads to the desired flexibility. In addition, with the separation of the control plane and the data plane, the policy enforcement, network reconfiguration and evolution are simplified. In SDN, a single logically centralized controller can control a distributed set of switches to provide better services to end-host applications. In most cases, such as with an OpenFlow enabled SDN, this is done via the use of dedicated controllers which manage the packet forwarding switches.

Naturally, there is a fair amount of inherent ambiguity here with regard to how feasible this is. Common problems include *a*) performance: how long it takes for a controller to handle a request; *b*) reliability: whether or not the system works in cases of link/node failures, and how resistant it is to such failure; and *c*) scalability: how feasible it is to implement on a wide scale.

One of the most studied open research problem in SDN is the controller placement problem. This is the problem of, given a network topology, determining how many controllers are needed, and where they should be placed. This problem is relevant because to effectively scale an SDN network, a larger number of controllers are required - and they need to be placed efficiently in order to satisfy the end goal of the system; which could be performance, balance, reachability, or a combination of several factors. Extensive research primarily focusing on performance, or minimizing propagation delay when deploying controllers in wide-area networks, had been done for controller placement problem. Heller et al. [1] introduced the controller placement problem that, given the locations (defined in IV-A) of nodes (forwarding elements), consists of choosing controller locations to maximize performance while minimizing the delay between nodes and controllers. While this suffices for most cases, in other situations it might be more advantageous to place controllers to minimize downtime, maximize resistance to network error, and so forth; as in the case of an enterprise network where simply having an available controller is more important than ensuring the minimum possible latency. However, this opens the door for further questions, such as how specifically, to go about doing this, and how well-compatible it is with existing metrics for controller placement.

Our motivation is to investigate how controllers can be placed in order to optimize fault-tolerance, rather than performance. Heller et al. [1] briefly mention a few techniques for this, namely placing controllers to maximize the number of nodes within a latency bound and measuring the distance to the i-th closest controller, but do not further investigate either. In this paper, we focus on two main questions: *a*) What is the trade-off between performance and reliability? That is, how far below optimal placement for performance do you need to be, in order to make the network adequately resilient against error? *b*) In general terms, how fault-tolerant is it possible to make an SDN?

In Section II, the current state of art is discussed. Section III represents the motivation and research issue. In Section IV, we define our fault-tolerance approach. Section V, and Section VI demonstrate the experimental steup and results, respectively. Finally, we summarize our findings and discuss future extensions to our analysis in Section VII.

II. Current State of Art

Distributed controller environments in SDN are widely studied. Onix [2] facilitates distributed control in SDN by providing each instance of the distributed controller access to holistic network state information through an API. HyperFlow [3] synchronizes the network state among the distributed controller instances while making them believe that they have control over the entire network. Kandoo [4] is a framework tailored for a hierarchical controller setup. It separates out local applications that can operate using the local state of a switch; and lets the root controller handle applications that require network-wide state. DISCO [5] is a distributed control plane that relies on a per domain organization, and contains an east-west interface that manages communication with other DISCO controllers. It is highly suitable for a hierarchically decentralized SDN controller environment. ONOS [6] is an OS that runs on multiple servers, each of which acts as the exclusive controller for a subset of switches and is responsible for propagating state changes between the switches it controls. Several of these work serve as essential groundwork for the controller decentralization strategies that serve as a necessary condition for the controller placement problem.

In [1], Heller et al. presented the controller placement problem and demonstrated the different scenarios of the controller placement by using real topologies [7] and Internet Topology Zoo [8]. However, the paper only provides an initial analysis of the controller placement problem and does not provide the technique to determine the optimal latency in SDNs. Dixit et al. [9] presented an approach to dynamically assign switches to the controllers for load balancing the controllers in real-time. The balanced placement of controllers can reduce the cost and the overhead for dynamic assignment of controllers. Moreover, dynamic assignment of controllers induces more radius stretch [10]. Bari et al. [11] also presented a technique to dynamically place controller depending on the changes of number of flows in the network. However, they did not consider optimizing the placement of controller when there is an outage of controllers.

Hu et al. [12] provide an approach to increase the elasticity of the SDNs when there is a failure of links between controllers and nodes. However, their approach was heuristic and the result was not satisfactory. Ros et al. [13] also presented a heuristic approach to increase the reliability of SDNs by using heuristic search on the number of controllers assigned to each node. In [14], Borococi et al. presented a study for final selection among several controllers using multi-criteria decision algorithm. However, the technique is not implemented for real life case studies using Internet Topology Zoo. Yao et al. [10] presented a capacitated controller placement problem by developing an effective algorithm by reducing the number of controllers and load on the controller. Xiao et al. [15] provide a new method to solve the controller placement problem with the consideration of the propagation latency in WANs. However, their proposed solution considers small parts of SDNs, what they called SDN-domains, as they partitioned the network to multiple SDN domains. While it shows latency improvement when using spectral clustering method, the improvement is mostly based on the size of each SDN domain. For the evaluation, the authors compare spectral clustering algorithm to optimize for average latency. While this work may obtain multiple SDN domains as its result, it needs to set the number of SDN domains (K) manually. Research on network reliability has been extensively conducted [16]. However, it is not related to SDN paradigm. Moreover, the most SDN related works on controller placement problems are directed to performance perspective [11], [8], [10], [7]. Few works have been done on controller placement with the consideration of resilience [17], [15], [18], [12], [19]. However, their focus only on resilience against network failures and do not consider any additional metrics [20].

III. Motivation

Science demilitarized zone (DMZ) has deployments from university campuses all across the United States. It is likely that most of these deployments have some associated compute and storage nodes on site. However, depending on the number of students or the load on the systems, there may not be a dedicated SDN controller at each location. Alternately, since controllers in an SDN environment are a single point of failure, targeted attacks on controllers might result in a controller failing. Hence, finding the optimal placement of the controllers such that researchers at every site can conduct experiments requiring high availability at high performance computing at all times requires solving the fault-tolerant controller placement problem. A solution to this could involve placement of an SDN controller at one or more of the Science DMZ sites to ensure resilience against controller failure from either adversarial or non-adversarial sources.

In [1], Heller, Sherwood, and McKeown investigate a wide range of topologies with respect to two metrics, average-case latency and worst-case latency. These two metrics are defined as follows.

Definition 1. *Given a weighted network graph $G = (V, E)$ with $n = |V|$, edge weights representing propagation delays, and a placement of controllers $S' \subseteq V$, the* ***average*** *propagation delay is:*

$$L_{avg}(S') = \frac{1}{n} \sum_{v \in V} \mathtt{min}_{s \in S'} d(v, s) \tag{1}$$

Definition 2. *Given a weighted network graph $G = (V, E)$ with $n = |V|$, edge weights representing propagation delays, and a placement of controllers $S' \subseteq V$, the* ***worst-case*** *propagation delay is:*

$$L_{wc}(S') = \mathtt{max}_{v \in V} \mathtt{min}_{s \in S'} d(v, s) \tag{2}$$

In both of these cases, the goal is to find a S' of given size k that is minimal with respect to either of $L_{avg}(S')$ or $L_{wc}(S')$. The problems of optimizing with respect to both, average and worst-case propagation delays has already been studied. The minimum k-median problem optimizes with respect to average-case latency, and is detailed in [21]. The corresponding optimization problem for worst-case latency is known as minimum k-center problem [17]. Although inefficient, for our research, it suffices to check every potential S' via brute force and pick the optimal one, hence this falls outside the scope of this paper. While [1] concludes that in most topologies, one controller is sufficient for most existing performance requirements, a focus on maximizing performance fails to take into account fault-tolerance. The immediate example is that placing just one controller is not adequate in a scenario where controllers may fail or otherwise temporarily be disabled, e.g. if the connection is lost or if the system it is running on must be taken down for maintenance. Another, more nuanced, example is a straight-line topology, say $G = (V, E)$ with constant edge weight ($w_G(e) = 1$; $\forall e \in E$), requiring two controllers. When planning for worst-case latency, the two controllers should be placed $1/4$ and $3/4$ down the line (allowing a worst-case latency of $|V|/4$), but this fails to take into account the possibility of a controller being unavailable (which would cause a potential worst-case latency of $3|V|/4$ for nodes at the "failing" end of the topology). In an error-critical system that cannot allow a single late transmission, it might thus be considered a safer option to place both controllers in the middle to allow a worst-case latency of $|V|/2$ even if one controller fails.

IV. Our Approach for Fault Tolerance

Our main approach is to handling controller placement as an optimization problem with regard to a predefined metric with one key difference: the metrics we use are aimed to measure the fault-tolerance of a system, as opposed to solely considering performance. Given a robust metric for fault-tolerance, the end result of this is that the optimal S' with regard to our metric results in a configuration of controllers that prioritizes fault-tolerance over performance. It is not quite as clear-cut what constitutes an error-resistant system as opposed to a well-performing system, as there are different ways in which networks can fail and varying effects depending on how critical the network is. With performance, the distinction is fairly straightforward: higher is better, lower is worse; but there are different ideas of what constitutes fault-tolerance in a system.

For purposes of our research, we distinguish between *error-critical* and *non-error-critical* systems. We consider an error-critical system to be a system where any transmission that is missed, or falls outside its defined performance requirements, causes a large adverse effect to the system and is thus unacceptable. Ergo, the system must be set up in a way where it can be ensured that every transmission succeeds and arrives on time. In other words, we must *deterministically* ensure that the latency never exceeds a certain amount. On the other hand, for a non-error-critical system, it suffices to make a best-effort attempt to minimize the overall latency of the system, allowing occasional failed or late transmissions. One immediate example is web traffic (most people won't care if a web page takes so much as five extra seconds to load, or if it doesn't load at all for whatever reason, as they can just try again and refresh the page). We consider metrics tailored to both kinds of systems. Now present a variety of potential fault-tolerance metrics we evaluated for our approach.

A. Model Assumptions

- Every controller c has the same non-zero chance of 'failing'. Failure can be due to any reasons, including hardware failure, software crashes and environmental reasons.
- Location in this work is contextual. In other words, what we refer to as location is a combination of three factors *a*) density, which describes how many switches to be controlled by the controller; *b*) usage, which defines how busy the controller is; and *c*) the geo-location. Expanding location to include these three factors, making location essentially 3-tuple Density-Usage-Location (DUL) gives us flexibility to enhance reliability by not only moving the geo-location around, but also change usage and density. However, to simplify evaluation, we assume fixed density and usage, thereby having geo-location as the only variable.

B. Maximum Cover

One well-established problem is the *maximum cover* problem [22], which is as follows: Given a collection of sets $S = \{S_1, S_2, \ldots, S_m\}$, where each $S_i \subseteq \{v_1, v_2, \ldots, v_n\}$, and a number k; identify a sub-collection $S' \subseteq S$ such that $|S'| = k$ and $|\cup_{S_i \in S'} S_i|$ is maximized. When applied as a networking metric, each S_i represents a node, and consists of the nodes within a latency bound to that node (passed as a parameter to the metric). This has the effect of maximizing the number of nodes within a certain latency bound. We chose not to further investigate this metric, as the scope of this paper is to find ways to ensure every node has acceptable reachability rather than maximizing the number of nodes that do so.

C. Number of Controllers within a Latency Bound

A similar approach is, given a pre-specified latency bound, we find an S' to maximize the minimum number of available controllers within the latency bound to each node. This is useful for determinism since we know for sure that a certain number of controllers could fail while still always having an available controller within an acceptable latency. We did not investigate this metric further, since we consider a wide range of topologies spread out over varying distances and fixing a one-size-fits-all latency bound is counterproductive. However, we imagine this would be more useful in the context of placing controllers for a single LAN as opposed to defining a catch-all metric.

D. Latency Bound for Closest i Controllers

The reverse of the above problem is, given the number of controllers k and a parameter i (representing the number of closest controllers we consider to each node), we find an S' to minimize the worst-case latency from each node to any of its nearest i controllers. This is equivalent to measuring the worst-case latency to the ith-nearest controller. This ensures the same condition (bounding the latency even in the case of i-1 controllers being dropped), but is more practical as bounding the number of controllers as opposed to enforcing a latency bound is more applicable to a generic topology.

E. α-adjusted Average-Case/Worst-Case Latency

Another approach is to directly model the possibility of error (that is to say, a controller node failing to function or being down for whatever reason). By doing so, it becomes possible to modify the existing average-case/worst-case metrics to factor error into account. Define a *rate of failure* $\alpha \in [0, 1]$. This is a parameter that determines the probability a controller, at any single given transmission, fails - any packet that would be forwarded to the controller must then go to a different controller instead (we operate under the assumption that this is the next-closest controller). This value is universal, given our assumption that every controller in S' has the same chance of failing. We then modify average-case and worst-case latency as follows:

$$L(v) = \mathtt{min}_{s \in S'} d(v, s) \tag{3}$$

$$L_{avg}(S') = \frac{1}{n} \sum_{v \in V} L(v) \tag{4}$$

$$L_{wc}(S') = \mathtt{max}_{v \in V} L(v) \tag{5}$$

These are equivalent to the original metrics - the key point here is that the function that must be minimized is abstracted. After this, we modify L as follows:

$$\begin{aligned} L(v, i) = \mathtt{min}_{s_1 \in S'} d(v, s_1) + \alpha \mathtt{min}_{s_2 \in S' \setminus \{s_1\}} d(v, s_2) + \\ \alpha^2 \mathtt{min}_{s_3 \in S' \setminus \{s_1, s_2\}} d(v, s_3) + \ldots + \\ \alpha^{i-1} \mathtt{min}_{s_i \in S' \setminus \{s_1, \ldots s_{i-1}\}} d(v, s_i) \end{aligned} \tag{6}$$

$$L^*_{avg}(S', i) = \frac{1}{n} \sum_{v \in V} L(v) \tag{7}$$

$$L^*_{wc}(S', i) = \mathtt{max}_{v \in V} L(v) \tag{8}$$

This is, in short, the adjusted latency for any given node when α is taken into account. It is probabilistic as opposed to deterministic, that is, it does not measure the latency of any given transmission but the average of all transmissions over time. To break down the formula, it measures the propagation delay to the first controller and then considers what happens if it fails (probability α). When this happens, we must retransmit to the second-closest controller, which accounts for a total of the average latency, and we must also consider what happens if this fails as well (probability α^2). This continues for the nearest i controllers, where i is an additional parameter to the metric. There are some notable properties about α-adjusted latency as defined here. First off, when $\alpha = 0$, both metrics are equivalent to the original two metrics established in [1]. Second, when $\alpha = 1$, it is equivalent to measuring the sum of either average-case or worst-case latency to the nearest i controllers. Since all metrics are scaled in this way, it has the effect of measuring the average propagation delay to the closest i controllers. Third, it is usually reasonable to set α higher than the actual rate of failure of the controllers - when α is exceptionally low, the α-adjusted metrics are almost equivalent to the original metrics. Since the end goal is not to obtain a perfectly-accurate measure of performance but to make the controller placement as resilient against error as possible, α does not have to be exactly what it is in practice.

V. Experimental Setup

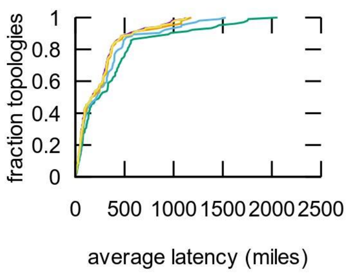
As stated in Section I, our research aimed to address two main questions: *a*) What is the trade-off between performance and reliability? That is, how far below optimal placement is required in order to make the network adequately resilient against error? *b*) In general terms, how fault-tolerant is it possible to make an SDN? In effect, given the metrics defined in Section III, we tested *a*) how much of a latency trade-off was required to place controllers according to our metrics; and *b*) the robustness of our metrics compared to the original metrics. We detail our methods for both experiments below.

A. Comparing Latency (COMPARE)

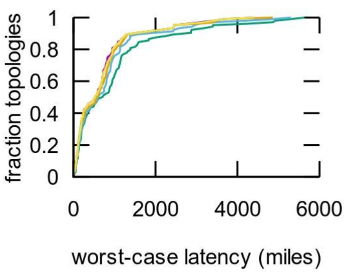
We analyzed controller placements on a wide range of topologies from the Internet Topology Zoo [8]. Utilizing the same code used in [1], we ignore outdated versions of topologies with multiple versions in order to avoid bias, and we remove ambiguous locations/edges when handling topologies that include them. We analyzed 106 topologies in total: 110 up-to-date topologies minus four (Latnet, Telcove, Uninett2010, Uninett2011) that we were unable to manage due to size and computational constraints. We tested optimal placements of $k = 4$ controllers for α-adjusted average-case and worst-case latency (trying $\alpha = 0.1, 0.25, 0.5, 1$) and worst-case latency to the nearest ith controller (trying $i = 2, 3$). We assumed at any given point that no more than two controllers would fail at any given transmission - we imagine most controllers in practice are reliable enough to ensure this enough of the time, and leave the extension to future work. As a result of this, we limited our calculations of α-adjusted latency to the nearest three controllers, and limited i to 3 when calculating the latency to the nearest ith controller. k was also limited to 4 due to computational constraints; at $k = 5$ we started running into memory errors because the size of the search space grows exponentially with regard to k. In general terms, the algorithm we used to implement COMPARE is shown in Algorithm 1.

Algorithm 1: COMPARE

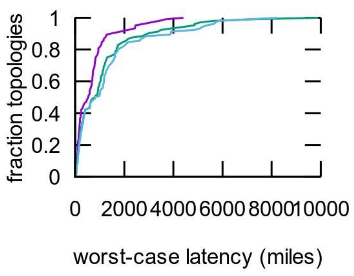
```
Input  : $(m, \alpha)$ or $(m, i)$ depending on metric
Output : average-case, worst-case latency
1 Procedure COMPARE()
2     for all topologies do
3         Compute optimal $S'$ for $k = 4$, metric $(m, \alpha)$ or
            $(m, i)$ on given topology
4         Compute average-case and worst-case latency for
            $S'$ on given topology.
```



(a) Average latencies when optimized for different values of α (left to right: normal, $0.1, 0.25, 0.5, 1.0$).

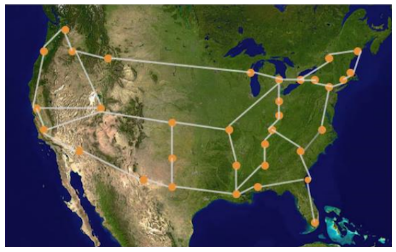


(b) Worst-case latencies when optimized for different values of α (left to right: normal, $0.1, 0.25, 0.5, 1.0$).

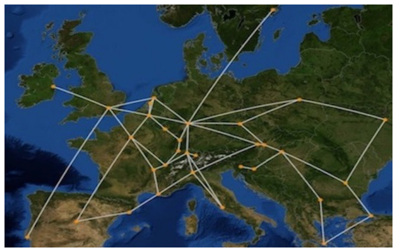


(c) Worst-case latencies when optimized for worst-case latency to the nearest ith controller (left to right: $i = 1, 2, 3$)

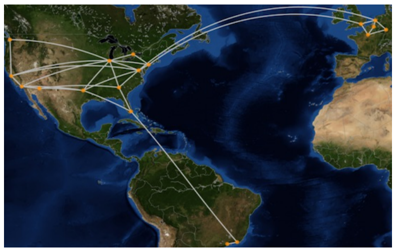
Fig. 1: Comparison of latencies across different topologies.



(a) OS3E11.



(b) BICS.



(c) Highwinds.

Fig. 2: Test topologies.

B. Robustness Tests (DROP)

In order to test robustness, we directly tested the propagation delay in scenarios where one or more controllers were deliberately dropped (simulating failure). Due to the relative complexity of DROP as opposed to COMPARE, we limited our analysis to three topologies: the Internet2 OS3E topology, as well as two outliers highlighted in [1] that were selected specifically for their shape (Highwinds, a medium-size topology with three distant 'islands' spanning three continents, and Bics, a medium-size mesh topology in Europe). These topologies are shown in Figure 2. The idea behind DROP is that we start with an optimal placement S' of k controllers, and then we drop some subset $S \subseteq S'$ of them, where $|S| = i$ (k and i both passed as parameters to the experiment). Then we measure the average-case and worst-case latencies using the updated set of working controllers, $S' \setminus S$. This has the effect of testing how resilient against error the resulting S' is.

Similar to COMPARE, we made the assumption that no more than two controllers would fail at any given transmission. Thus we tested values 1 and 2 for i. Since the idea behind the experiment was to limit the number of available controllers, we used larger values of $k (k = 5, 6)$, which in this case was possible computationally since none of the three topologies we tested were large in size. Due to the limited search space, we tested every single potential subset S of failed controllers, and measured the average latencies across all possible combinations. We tested α-adjusted average-case and worst-case latency with $\alpha = 0.5$ and worst-case latency to the 3rd-nearest controller against the original average-case and worst-case latency metrics. Since the purpose was to test if the metric worked correctly or not, we saw it as redundant to test alternate values of α and i, instead just picking arbitrary values near the middle for both. The algorithm we used to implement DROP is shown in Algorithm 2.

Algorithm 2: DROP

```
  Input  : $(m, \alpha)$ or $(m, i)$ depending on metric
  Output : average-case, worst-case latency
1 Procedure DROP()
2     Set $S'$ = the optimal placement of $k$ controllers on
        $topo$ with respect to $m$
3     for all subsets $D$ of $S'$ of size $i$ do
4         Set $S = S' \setminus D$.
5         Compute optimal $S'$ for $k = 4$, metric $(m, \alpha)$ or
            $(m, i)$ on given topology
6         Compute worst-case latency of placement $S$ on
            $topo$
7         After checking every combination, compute
            average and standard deviation of all worst-case
            latencies.
```

VI. Results

A. COMPARE

For $k = 4$, placing controllers optimally with respect to α-adjusted average case latency resulted in latency increases that were approximately proportional to the optimized average-case latency placement. For $\alpha = 0.1$ and $\alpha = 0.25$, the resulting latency appeared to be almost identical; there was a slight increase at $\alpha = 0.5$, and at $\alpha = 1.0$ the latency maxed out at approximately $1.5 - 1.7$x the standard average-case latency. Placing controllers with respect to α-adjusted worst-case latency yielded similar results, without as much of an increase - the latency at $\alpha = 1.0$ was generally about $1.2-1.3$x as much as the standard worst-case latency. Figure 1 shows the CDFs for the α-adjusted average/worst-case latencies on all 106 topologies, demonstrating the proportional increase. The trend pretty steadily moves upward the more α (or i) is increased. Placing controllers with respect to the worst-case latency bound on the ith nearest controller yielded a larger increase relative to standard worst-case latency (over 2.0x on topologies with greater average distance). The CDF is shown in Figure 1.

B. DROP

In general, for $k = 5$, the fault-tolerance optimized metrics performed comparably to the standard metrics, leading us to conclude that considering fault-tolerance metrics did not lead to a drop in performance. Figure 3 shows the histograms for the average worst-case latency when computed across all possible sets S of dropped controllers on the OS3E, Bics, and Highlands topologies (the bars represent average-case latency, worst-case latency, average-case latency with $\alpha = 0.5$, worst-case latency with $\alpha = 0.5$, and worst-case latency to the ith nearest controller with $i = 3$ respectively).

For $k = 6$, there is some evidence to suggest that at least one fault-tolerant specialized metric performed better than average. In OS3E for $(k, i) = (6, 1)$, α-adjusted average case latency in miles was 79 miles less than the second place metric (1016 miles compared to α-adjusted worst-case latency's 1095 miles). $(k, i) = (6, 2)$ looked more promising with 3rd-worst-case latency noticeably decreasing as well. While some fault-tolerant metrics placed better on Bics and Highlands as well, there did not appear to be much of a pattern in specifically which metrics were most effective across different topologies. This indicates that no one of the metrics we investigated was particularly well-suited for all topologies and that more research should be done in order to develop a stronger fault-tolerant metric. One factor that likely played into these results, Highlands in particular, is that one of the 'islands' on Highlands only contains 2 nodes. This explains why optimizing for α-adjusted latency performed better there, as it was more prone to putting controllers on this island - and not optimizing for 3rd-worst-case latency (in which case it's impossible to put 3 controllers close to either node on said island). It is unclear whether this trend will continue for larger values of k we were not able to test this because of computational limits but these results are at least some indicator that for large topologies with many controllers, it may be worth investigating to determine if there exists a more effective metric that does not sacrifice so much latency on average. For the time being, though, it is likely not worth it to sacrifice the extra latency these placements were not much better than random in most cases, which usually performs at a factor of around 2.0x [1]. Interestingly, it did not appear that i-the number of controllers dropped had any effect on anything. We had hypothesized that the more errors occurred, the greater the advantage the fault-tolerant metrics would have over the latency-optimized metrics.

(k, i)	Sample standard deviation (miles)		
	Worst-case	α-adj. worst-case, $\alpha = 0.5$	3rd-nearest
$(5, 1)$	294.681	337.369	203.234
$(5, 2)$	382.534	282.416	190.361
$(6, 1)$	146.319	281.427	42.213
$(6, 2)$	208.216	249.779	88.663

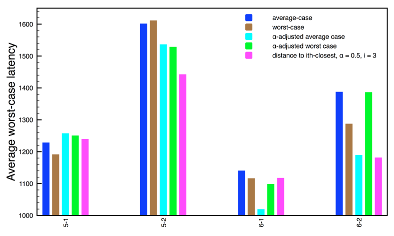
TABLE I: Sample standard deviations for worst-case metrics when running DROP on OS3E topology. In all four cases, the worst-case latency to the 3rd-nearest node had a noticeably lower standard deviation.

C. Standard Deviation of 3rd-Worst-Case Latency

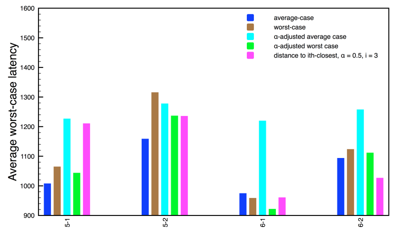
The other interesting result during DROP which we did not intend to happen at all when we initially designed it was that the standard deviation of the recorded worst-case latencies when optimizing for 3rd-worst-case latency was noticeably lower than that of the other metrics, both those measuring average-case and worst-case. The results for the OS3E topology versus the worst-case metrics are listed in Table I; the other topologies had similar results. This makes sense, since the purpose of 3rd-worst-case latency is indeed to be predictable.

VII. Conclusion and Future Work

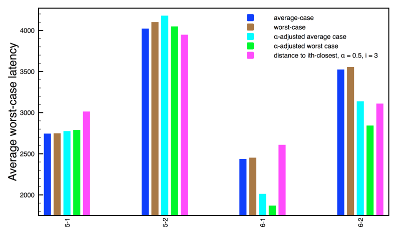
We consider the significant problem of ensuring reliability and availability of the network, and propose a solution for fault-tolerant controller placement. We analyzed our work on a wide range of topologies from the Internet Topology Zoo, and evaluated the effect of the number of available controllers. For small values of k, it is likely not worth it to optimize for fault-tolerance over latency using these metrics. The increase in latency is simply too high. However, as k increases, there is evidence to suggest it may be worth it to balance around reachability as opposed to assigning controllers purely with regard for latency. It definitely would not be worth it in a system where it could be reasonably assumed that no major errors would occur. If one does choose to optimize for fault-tolerance, it is likely that minimizing the distance to the ith-closest controller is a promising compromise, as it appeared to be remarkably consistent even in situations where multiple controllers failed. This is, of course, not to say that no such metric exists that can provide an adequate amount of fault-tolerance while maintaining low latency. It just means that



(a) OS3E



(b) Bics



(c) Highlands

Fig. 3: Histogram measuring the average worst-case latency (miles) for different topologies when i/k controllers are dropped across all metrics.

none of the metrics we tried in this specific experiment were fully effective. This experiment was pretty severely limited by a lack of computational power as alluded to multiple times, we could not continue COMPARE anywhere past $k = 4$. More research in multiple areas would clear a lot of things up, in particular, the issue of whether fault-tolerant metrics continue gaining ground as k increases. More research in metrics is also required, since there did not appear to be any kind of set pattern in which metrics performed better and why, indicating that an optimal metric will likely combine elements from several other metrics to fall in between. In our future work, we will consider weighted averaging approach that is, given a controller placement S', for each node, calculate a weighted average of the propagation delay to each $v \in S'$. This would have the effect of giving a bonus whenever a node has several controllers placed closely to it, while minimizing the effect of faraway controllers.

Acknowledgments

This research is based upon work supported by the NSF under Grants 1642031 and 1528099, and NSFC under Grants 61628201 and 61571375.

References

[1] Brandon Heller, Rob Sherwood, and Nick McKeown. The controller placement problem. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 7–12. ACM, 2012.

[2] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, et al. Onix: A distributed control platform for large-scale production networks. In *OSDI*, volume 10, pages 1–6, 2010.

[3] Amin Tootoonchian and Yashar Ganjali. Hyperflow: A distributed control plane for openflow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, pages 3–3, 2010.

[4] Soheil Hassas Yeganeh and Yashar Ganjali. Kandoo: a framework for efficient and scalable offloading of control applications. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 19–24. ACM, 2012.

[5] Kévin Phemius, Mathieu Bouet, and Jérémie Leguay. Disco: Distributed sdn controllers in a multi-domain environment. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–2. IEEE, 2014.

[6] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, et al. Onos: towards an open, distributed sdn os. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6. ACM, 2014.

[7] Soheil Hassas Yeganeh, Amin Tootoonchian, and Yashar Ganjali. On scalability of software-defined networking. *Communications magazine, IEEE*, 51(2):136–141, 2013.

[8] Simon Knight, Hung X Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775, 2011.

[9] Advait Dixit, Fang Hao, Sarit Mukherjee, TV Lakshman, and Ramana Kompella. Towards an elastic distributed sdn controller. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 7–12. ACM, 2013.

[10] Guang Yao, Jun Bi, Yuliang Li, and Luyi Guo. On the capacitated controller placement problem in software defined networks. *IEEE Communications Letters*, 18(8):1339–1342, 2014.

[11] Md Faizul Bari, Arup Raton Roy, Shihabur Rahman Chowdhury, Qi Zhang, Mohamed Faten Zhani, Reaz Ahmed, and Raouf Boutaba. Dynamic controller provisioning in software defined networks. In *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, pages 18–25. IEEE, 2013.

[12] Yannan Hu, Wendong Wang, Xiangyang Gong, Xirong Que, and Cheng Shiduan. On the placement of controllers in software-defined networks. *The Journal of China Universities of Posts and Telecommunications*, 19:92–171, 2012.

[13] Francisco Javier Ros and Pedro Miguel Ruiz. Five nines of southbound reliability in software-defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 31–36. ACM, 2014.

[14] Eugen Borcoci, Radu Badea, Serban Georgica Obreja, and Marius Vochin. On multi-controller placement optimization in software defined networking-based wans. *ICN 2015*, page 273, 2015.

[15] Peng Xiao, Wenyu Qu, Heng Qi, Zhiyang Li, and Yujie Xu. The sdn controller placement problem for wan. In *2014 IEEE/CIC International Conference on Communications in China (ICCC)*, pages 220–224. IEEE, 2014.

[16] Michael O Ball, Charles J Colbourn, and J Scott Provan. Network reliability. *Handbooks in operations research and management science*, 7:673–762, 1995.

[17] Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[18] Neda Beheshti and Ying Zhang. Fast failover for control traffic in software-defined networks. In *Global Communications Conference (GLOBECOM), 2012 IEEE*, pages 2665–2670. IEEE, 2012.

[19] Yannan Hu, Wendong Wang, Xiangyang Gong, Xirong Que, and Cheng Shiduan. Reliability-aware controller placement for software-defined networks. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 672–675. IEEE, 2013.

[20] Stanislav Lange, Steffen Gebert, Thomas Zinner, Phuoc Tran-Gia, David Hock, Michael Jarschel, and Marco Hoffmann. Heuristic approaches to the controller placement problem in large scale sdn networks. *IEEE Transactions on Network and Service Management*, 12(1):4–17, 2015.

[21] Michael Shindler. Approximation algorithms for the metric k-median problem. *Written Qualifying Exam Paper, University of California, Los Angeles. Cited on*, page 44, 2008.

[22] Dorit S Hochba. Approximation algorithms for np-hard problems. *ACM SIGACT News*, 28(2):40–52, 1997.