SafeGuard: Congestion and Memory-aware Failure Recovery in SD-WAN

Meysam Shojaee
Faculty of Computer Science
Dalhousie University
meysam.shojaee@dal.ca

Miguel Neves
Institute of Informatics
UFRGS
mcneves@inf.ufrgs.br

Israat Haque
Faculty of Computer Science
Dalhousie University
israat@dal.ca

***Abstract*—In software-defined WANs (SD-WAN), link failure can lead to congestion and packet loss, hence degrading application performance. State-of-the-art traffic engineering approaches can speed up failure recovery by proactively installing backup tunnels and redirecting affected traffic immediately in the data plane, which reduces the burden on the network controller. However, these approaches either lead to bandwidth waste because of reserved link capacity or impose restrictions on network topologies, e.g., the existence of link-disjoint routes or large switch memory resources. In this paper, we propose *SafeGuard*, a software-defined proactive recovery system that improves bandwidth allocation and switch-memory usage while working on any connected network. We formulate the failure recovery problem as a multi-objective MILP optimization problem for all possible single link failures, the most common case in current WANs as temporally-coinciding failures are rare. We then develop a heuristic to efficiently compute backup routes as the problem is NP-Hard. We implemented a prototype of SafeGuard using the Ryu SDN controller and extensively evaluate it in Mininet over two real topologies, Google B4 and ATT. Our results show that SafeGuard can reduce the number of congested links by up to 50% compared to the state-of-the-art failure recovery scheme.**

***Index Terms*—Software-defined networking, SD-WAN, failure recovery, multi-objective optimization.**

I. INTRODUCTION

Software-Defined Networking (SDN) [1], [2] enables network programmability that leads to flexible network configuration and rapid innovation. In SDN, failures can occur in the management, control, or data plane [3]. In this work, we focus on data-plane failures, specifically communication link ones. An extensive study on Microsoft's WAN reveals that the probability of having a single link failure in a five-minute interval is above 20%, which has a notable impact on link utilization [4].

One of the most common ways to recover from failures in SD-WANs is through Traffic Engineering (TE). In particular, SDNs leverage dynamic route selection and bandwidth allocation to offer a balance between network availability and efficient resource (i.e., link capacity and switch memory) utilization. There are mainly two failure recovery approaches in SDN: *reactive* and *proactive* [3]. In the reactive approach, the controller responds to a link failure upon receiving a notification from the data plane. Thus, the communication between the affected switches and the network controller, the computation of a new route, and the device configuration introduce overhead and delay to recover from a failure [5]. On the other hand, the proactive recovery scheme eliminates such overhead and delays by installing backup routes at switches to locally recover from a failure without the controller's intervention. Nevertheless, proactive approaches usually require extra switch memory space to store backup routes, which can lead to contention on this scarce resource.

Previous work on the state-of-the-art has adopted variations of proactive recovery schemes to route traffic with reduced congestion in the presence of failures. For example, Forward Fault Correction (FFC) [6] is resilient up to k concurrent failures at the cost of keeping some spare capacity. Traffic is then rerouted through spare links in case of a failure. Sentinel [7], on the other hand, does not require any spare capacity at all. Instead, it installs backup routes and solves an ILP model to reroute traffic while minimizing the maximum link utilization. However, these solutions have two important drawbacks: i) they require special topologies (e.g., containing link-disjoint paths) to work properly; and ii) they miss incorporating switch memory constraints while selecting routes and assigning traffic rates to them. Ultimately, exhausted switch memory can lead to packet drops and increase table lookup and update times [8].

In this paper, we propose a novel software-defined failure recovery system for SD-WANs. Our system, called *SafeGuard*, takes into account both link and switch memory constraints for proactively allocating backup routes. We formulate the failure recovery problem as a multi-objective MILP optimization problem that selects routes (either primary or backup) and assigns rates to them for all possible single link failures (the most common case in WANs [7]). Unlike prior work, the proposed model is not confined to any specific network topology and can encompass different route selection algorithms (e.g., ECMP, k-shortest, or link-disjoint paths) while minimizing both route length and link utilization. As our model turns out to be NP-hard, we develop a heuristic that greedily selects routes and assigns rates to them. Our heuristic is enhanced to find a balance between link and switch memory utilization and to avoid resource waste due to fragmentation (i.e., underutilizing bandwidth resources because of quickly exhausting flow table entries or vice-versa).

We implemented a prototype of SafeGuard using the Ryu

978-3-903176-31-7 © 2020 Crown

SDN controller, which incorporates our heuristic and uses OpenFlow Fast Failover Groups to immediately reroute traffic when a failure happens. Our code is publicly available [9]. We perform an extensive evaluation of SafeGuard in Mininet [10] over two real topologies, Goggle WAN (B4) and ATT, and compare it with Sentinel [7], the state-of-the-art SD-WAN failure recovery scheme. Our results show that SafeGuard can reduce the number of congested links and backup route length by up to 50% and 15%, respectively. In summary, we make the following contributions:

- We formulate the failure recovery problem in SD-WANs as a multi-objective optimization problem and solve it using CPLEX. Unlike previous work, our model considers both link and switch memory constraints and enables operators to consider multiple objectives altogether.
- We develop a greedy heuristic to solve our model in a reasonable time and design a software-defined failure recovery system, called SafeGuard, to deploy our heuristic on large scale SD-WANs.
- We implement a prototype of SafeGuard using the Ryu SDN controller and make our code publicly available [9].
- We perform an extensive evaluation of SafeGuard over two real WAN topologies and show it can efficiently utilize the link and switch memory resources while protecting against all possible single link failures.

The rest of the paper is organized as follows. We present our multi-objective optimization model and describe the corresponding heuristic in Sections II and III, respectively. Section IV describes the SafeGuard system. Section V provides a discussion on the evaluation results. Finally, we present the related work in Section VI and conclude the paper in Section VII.

II. PROBLEM FORMULATION

In this section, we formulate the failure recovery problem in SD-WANs as a MILP optimization problem. We elaborate on the proposed model in the next subsections.

A. Network Design

We present a network as a digraph $G = (V, E)$, where $V = \{1, 2, .., n\}$ is the set of network switches (or nodes) and E is the set of links among the switches. Each link $(i, j) \in E$ has a capacity c_{ij}. Each flow demand in this network is f and has a required bandwidth, b_f. We assume demands represent the *aggregate* traffic from an ingress to an egress switch and are split among a set of primary routes P_f. The traffic rate of flow f on each route $p \in P_f$ is referred to as t_{fp}[1]. In line with the literature [11], [7], our model considers a single link failure e [2]. In this case, the failure impacts all flows traversing that link. For each impacted flow f, we assume a set Q_f of backup routes is available. Each route $p \in Q_f$ uses a portion of the impacted

[1]For simplicity, we use p to denote any route in our model (either primary or backup).

[2]To consider all possible single link failures, one can repeatedly solve the model for different failing links. This makes the model easier to solve and allows instances for different failing links to be run in parallel.

TABLE I: Key notations used in the model.

	Notation	Description
Input	$G(V, E)$	Network graph with switches V and Links E;
	P_f	Set of primary routes for flow f;
	Q_f	Set of backup routes for flow f;
	t_{fp}	Traffic rate of flow f on route p;
	b_f	Bandwidth required for flow f;
	c_{ij}	Total capacity of link $(i, j) \in E$;
	Tc_i	TCAM space capacity of node $i \in V$;
[c]Auxiliaryvariables	e	A failed link;
	F_e	Set of impacted flows by link failure e;
	F	Set of flows not impacted by link failure e;
	$P_f^{'}$	Set of primary routes for flow $f \in F_e$ not impacted by link failure e;
	l_p	Length of route $p \in P_f^{'} \bigcup Q_f$;
	$r_{ij,p}$	1 if link $(i, j) \in p$, 0 otherwise;
	$s_{i,p}$	1 if node $i \in p$, 0 otherwise;
Output	x_{fp}	Allocation of affected flow f on route $p \in P_f^{'} \bigcup Q_f$;
	y_{fp}	Traffic rate of affected flow f assigned to route p;

primary route (i.e., from ingress switch to failing switch[3]), and an alternative path from the failing switch to the egress switch of the respective flow. Note that a link failure could impact one or multiple primary routes of a flow, as primary routes are not necessarily link-disjoint. We denote the set of non-affected primary routes as $P_f^{'}$. In either case, we select both backup and non-affected primary routes to compute a new routing for flow f after a failure. Table I summarizes these notations.

B. Variables and parameters

We need to assign traffic rates on backup and unaffected primary routes for each flow impacted by link failure e. For that, we define two sets of decision variables: x_{fp}, which is a binary decision variable denoting the routes used for impacted flow f after recovery, such that

$$x_{fp} = \begin{cases} 1 & \text{if } route\ p \in P_f^{'} \bigcup Q_f\ is\ used\ to\ forward \\ & traffic\ from\ flow\ f, \\ 0 & otherwise. \end{cases} \tag{1}$$

and y_{fp} to denote the traffic rate of impacted flow f assigned to route p after the failure. We also define binary variables, $r_{ij,p}$ and $s_{i,p}$, to identify the links and nodes belonging to route p, respectively:

$$r_{ij,p} = \begin{cases} 1 & \text{if } \ (i, j) \in p, \\ 0 & otherwise. \end{cases} \tag{2}$$

$$s_{i,p} = \begin{cases} 1 & \text{if } \ i \in p, \\ 0 & otherwise. \end{cases} \tag{3}$$

C. Constraints

1) Link and node capacity: constraints in (4) prevent link utilization from exceeding its maximum capacity. The utilization is calculated by summing up traffic rates assigned to both affected ($f \in F_e$) and unaffected ($f \in F$) flows, either for primary or backup routes. Constraints in (5), on the other hand, prevent switch memory usage from exceeding

[3]We define a failing switch as the source switch of a failing link.

its capacity. Similar to link capacity constraints, we calculate switch memory usage by summing (unitary) demands from primary and backup routes of both affected and unaffected flows. Note that $r_{ij,p} = 1$ implies node i belongs to route p of flow f.

$$\sum_{f \in F_e} \sum_{p \in P_f^{'} \bigcup Q_f} y_{fp} r_{ij,p} + \sum_{f \in F} \sum_{p \in P_f} r_{ij,p} t_{fp} \leq c_{ij} \quad \forall (i,j) \in E \tag{4}$$

$$\sum_{f \in F_e} \sum_{p \in P_f^{'} \bigcup Q_f} x_{fp} s_{i,p} + \sum_{f \in F} \sum_{p \in P_f} s_{i,p} \leq Tc_i \quad \forall i \in V \tag{5}$$

2) Traffic rate: constraints in (6) correlate variables x_{fp} and y_{fp}. They ensure that traffic rate of flow f on route p is zero when p is not used by f (i.e., $x_{fp} = 0$). Otherwise, it must be a positive number.

$$x_{fp} \leq y_{fp} \leq x_{fp} b_f, \quad \forall f \in F_e, p \in P_f^{'} \bigcup Q_f \tag{6}$$

3) Flow satisfaction: constraints in (7) ensure flow demands are satisfied, i.e., the total traffic rate of flow on its primary and backup routes is equal to its demand. Note we only need to consider constraints for affected flows (i.e., $f \in F_e$) as non-affected ones are satisfied by default.

$$\sum_{p \in P_f^{'} \bigcup Q_f} y_{fp} = b_f \quad \forall f \in F_e \tag{7}$$

D. Objective function

The objective function (8) consists of two terms. The first term optimizes the backup route length, while the second one reflects the overall link utilization in the network. The function also has two parameters, α and β, to favor a specific objective depending on the operator needs (e.g., to favor shorter routes due to application latency constraints or lower link utilization due to network congestion). We conducted an extensive analysis to empirically determine proper values for both parameters and concluded that α needs to be approximately 10x smaller than β to convey the same importance for both objectives[4].

$$\min \sum_{f \in F_e} \sum_{p \in P_f^{'} \bigcup Q_f} (\alpha x_{fp} l_p + \beta \sum_{(i,j) \in E} \frac{r_{ijp} y_{fp}}{c_{ij}}) \tag{8}$$

As a MILP problem, our model is known to be NP-Hard [12] and thus can take a long time to be solved in practice. For example, it took us more than an hour to solve this model for Cogent[5] topology on CPLEX. As traffic engineering intervals are usually below 5 minutes, we develop a heuristic to solve the SD-WAN failure recovery problem in a reasonable amount of time. We describe our heuristic in the next section.

[4]See [9] for more information.

[5]https://cogentco.com/en/network/network-map

Algorithm 1: Heuristic for rerouting traffic

Input : Network topology $G' = (V, E \backslash e)$; impacted flows F_e; set of available routes $P = P_f^{'} \bigcup Q_f, \forall f \in F_e$

Output: Traffic rates assigned to y_{pf}

```
1  Sort F_e in descending order according to flow demands
2  Sort P in ascending order according to route length
3  for f ∈ F_e do
4      for p ∈ P'_f ⋃ Q_f do
5          if ∀s ∈ p, u_s < Tc_s − 1 then
6              y_pf = min( min_{(i,j)∈p} a_ij, d_f )
7          else
8              if d_f ≤ min_{(i,j)∈p} a_ij then
9                  y_pf = d_f
10             end
11         end
12         if y_pf > 0 then
13             Update u_s, a_ij ∀s, (i,j) ∈ p
14             d_f ← d_f − y_pf
15         end
16     end
17 end
```

III. Proposed Heuristic

In this section, we present our proposed heuristic, along with an operational example. Algorithm 1 shows the procedure. It takes as input the network topology after removing the failed link e, the set of affected flows F_e, and a set of candidate routes P containing both backup routes and non-affected primary ones for all affected flows. Our goal is to reroute these flows over the candidate routes to satisfy their demands d_f within the link and switch memory budget. We start by sorting the affected flows and available paths in descending and ascending order according to their demands and route length, respectively (lines 1-2). Thus, we try to use the shortest routes first while greedily allocating the flow demands from biggest to smallest to avoid fragmentation issues. We then iterate over all candidate paths for a given flow, checking whether all switches in that path have enough *memory* capacity (u_s) for hosting the new demand (lines 3-5). When enough memory is available, we allocate the minimum between the flow demand and the remaining *bandwidth* capacity (a_{ij}) of the bottleneck link (line 6).

We adopt a "biggest demand first" strategy until we reach a given threshold in terms of switch memory utilization on a route ($Tc_s - 1$ for the bottleneck switch in our case)[6]. After that, we start looking for a "best-fit" demand for it, i.e., demands that can be fully satisfied to ensure that smaller flows have access to shorter routes too (lines 8-10). Finally, we update the remaining capacities and demand to be allocated whenever we select a path for a flow (lines 12-15). Although this approach can be

[6]Exploring different thresholds is possible, and we leave that as future work.

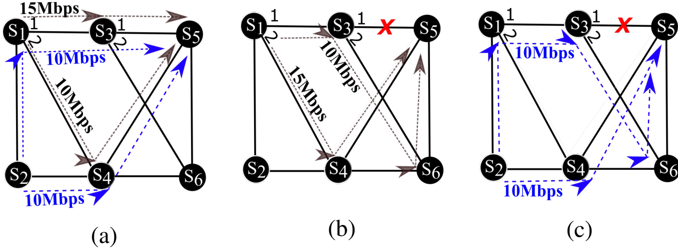


Fig. 1: a) Traffic rates on the primary routes of two flows (blue and black). The available capacity of all links is 20Mbps except link $S_4 - S_5$ whose capacity is 5Mbps; b) the available tunnels for the affected flow with the highest demand and corresponding traffic rates assigned to them; c) the available tunnels for the second affected flow after the link failure with the traffic rates assigned to them.

detrimental for some shorter flows, it is in line with our multi-objective optimization model, where we strike for a balance between *route length and link utilization*. In particular, purely allocating smaller demands first would lead to bigger demands being assigned to longer routes and thus to a higher number of more utilized links. The time complexity of Algorithm 1 is proportional to the number of flows affected by a link failure and the corresponding set of available routes. Suppose $|F_e| = M$ and $|P| = N$, then the time complexity of the algorithm is $O(MN(|V|+|V|log|V|))$, which is given by the operations at lines 3-6.

We illustrate the operation of Algorithm 1 in Fig. 1. There are two flows $\{f_1, f_2\}$: f_1 from switch S_1 to S_5 has a demand of 25Mbps while f_2 from switch S_2 to S_5 requires 20Mbps. Their routing tunnels and corresponding rates are shown in Fig. 1(a). For simplicity, we assume that all switches have enough memory to accommodate these two flow rules. We also assume that the available capacity of all links is 20Mbps except link $S_4 - S_5$. Its available capacity is 5Mbps; thus, it is the bottleneck link. The failure of link $S_3 - S_5$ impacts both $\{f_1, f_2\}$. In the following, we illustrate the operation of the heuristic.

Algorithm 1 starts with flow f_1 (black flow) as it has the higher demand. This flow has two available tunnels to send the traffic after the failure: $S_1 - S_4 - S_5$ and $S_1 - S_3 - S_6 - S_5$. It sends 15Mbps over the former tunnel. Note that before the failure, this tunnel could carry 10Mbps for f_1 because of sharing its capacity with f_2, which is not the case after the failure. The remaining 10Mbps is allocated over the second tunnel. On the other hand, the available tunnels for the second flow f_2 are: $S_2 - S_4 - S_5$ and $S_2 - S_1 - S_3 - S_6 - S_5$. The traffic allocation on the former one (10Mbps) does not change as it has no more capacity. The remaining 10Mbps can be sent over the second tunnel. Note that in both cases, one of the original tunnels remains unchanged except the corresponding rate that depends on the available capacity of the bottleneck link.

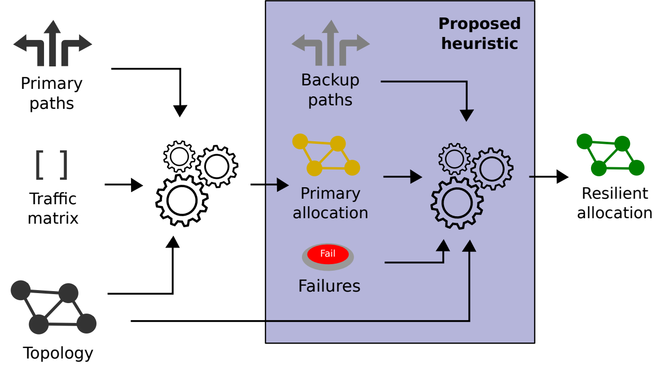


Fig. 2: SafeGuard architecture.

IV. SafeGuard overview

SafeGuard is an SDN failure recovery system that enables fast recovery while taking network congestion and scarce switch memory resources into account. SafeGuard runs as a controller application that deploys our proposed heuristic to derive resilient routing configurations to network switches at every traffic engineering interval.

Control plane. Fig. 2 shows SafeGuard architecture. It receives a traffic matrix, the network topology, and a set of forwarding routes (primary routes) as input, and produces a primary allocation that will be used for forwarding traffic under normal conditions (i.e., when there is no failure in place). SafeGuard then applies our proposed heuristic to augment the primary allocation with backup routes for all flows for each possible single link failure. Ultimately, it outputs a resilient routing allocation as a set of forwarding rules for SDN switches.

Data plane. SafeGuard uses the OpenFlow protocol (version 1.3 [13]) to manage network switches. Each switch includes at least one *Flow Table* and a *Group Table*. The flow table contains a set of flow rules, each consisting of match fields and corresponding action (or instruction). The group table, on the other hand, consists of group entries containing one or more action buckets. Table II shows an example. SafeGuard uses two types of groups: `select` and `fast failover` groups. The former groups assign traffic rates and balance load among multiple paths. This group type executes one action bucket in the group at a time according to a weight set as a parameter. The `fast failover` groups reroute traffic upon detecting a link failure (a down port), i.e., it executes the first live bucket.

Let us now refer to Fig. 1 and consider its first flow (f_1: from S_1 to S_5) to show how SafeGuard configures the flow and group tables at switches S_1 and S_3 to recover from a failure. Table II outlines the flow and group tables at the source switch S_1. We have one flow entry that points to one of the groups. Before the link failure, the flow entry points to group 1 (G1.1) of type `select` in the group table to assign traffic rates to the two available routes (see Fig. 1(a)). Group entry 1, on its turn, points to groups 2 and 3, which are of type `fast failover`. As both ports of S_1 are up in our example, G 1.2 and G 1.3 forward packets over ports 1 and 2, respectively.

TABLE II: Flow and group tables at S_1 for the example in Fig. 1

(a) Flow table at S_1

Match Field		
SrcAdrr	*DstAdrr*	*Instruction*
S_1	S_5	$G1.1$

(b) Group table at S_1

Group ID	Group Type	Action Buckets
G 1.1	select	weight: $\frac{3}{5}$, action: G 1.2 weight: $\frac{2}{5}$, action: G 1.3
G 1.2	fast failover	outport: 1 outport: 2
G 1.3	fast failover	outport: 2 outport: 1
G 1.4	select	weight: $\frac{2}{5}$, outport: 1 weight: $\frac{3}{5}$, outport: 2

TABLE III: Flow and group tables at S_3 for the example in Fig. 1

(a) Flow table at S_3

Match Field		
SrcAdrr	*DstAdrr*	*Instruction*
S_1	S_5	G 3.1

(b) Group table at S_3

Group ID	Group Type	Action Buckets
G 3.1	fast failover	outport: 1 outport: 2

Table III shows the flow and group tables at switch S_3, which detects the failure. Before the link failure, it sends packets from S_1 to S_5 through port 1 (route $S_1 - S_3 - S_5$). After detecting the failure, S_3 notifies the controller and immediately starts sending packets from S_1 to S_5 using port 2 because of the `fast failover` group (route $S_1 - S_3 - S_6 - S_5$). After receiving the failure notification from S_3, SafeGuard modifies the instruction in the flow table of S_1 to point to group 4 (G1.4) to update the allocated rates for each route used by the affected flow.

Workflow. Fig. 3 shows the SafeGuard workflow. The same process repeats every traffic engineering interval. First, SafeGuard computes and installs all primary tunnels and splitting weights for every flow (step A). Then, it proactively installs backup routes and computes the weights for allocated flows for allowing them to deal with any single link failure (step B). When a failure happens, the failing switch activates the corresponding backup tunnels connecting itself to the egress switches of the corresponding affected flows and sends a message reporting the failure to the network controller (step C). Finally, the network controller (which implements the SafeGuard heuristic) adjusts splitting weights for all affected flows at their respective ingress switches (step D).

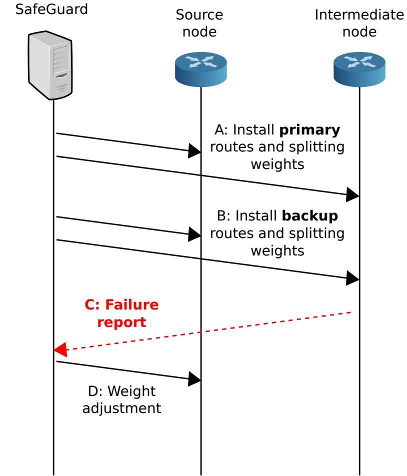


Fig. 3: SafeGuard workflow.

V. Evaluation

We implement SafeGuard as a Python application ($\approx$ 700 lines of code) on top of the Ryu SDN controller (version 4.30). The source code is available at [9]. In this section, we present our evaluation for the SafeGuard prototype.

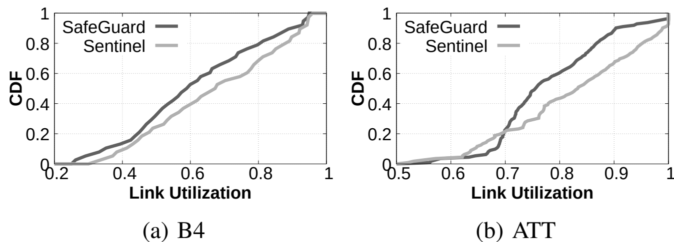
A. Setup

Our experiments run on a machine with 2.66GHz 12 core CPU and 44GB RAM equipped with Mininet (version 2.2.2). We consider two network topologies: Google B4 [14] (12 nodes and 38 links) and ATT [15] (25 nodes and 112 links). Each node is deployed as a CPqD[7] switch instance [16]. Link capacities are set to 1 Gbps with a 1ms delay. We benchmark SafeGuard against Sentinel [7], the state-of-the-art SD-WAN failure recovery approach. When a failure happens, Sentinel relies on the link-disjoint tunnels to distribute the traffic rate of affected flows. Sentinel targets minimizing the maximum link utilization while not considering the length of the backup tunnels or switch memory usage. The workload is comprised of UDP flows generated using *iperf*. Unless explicitly mention, we fix the flow rate (50 Mbps) and vary the number of flows in each experiment, similar to [7]. We use byte counters in the switches to get link utilization and measure the route stretch by looking at the selected backup routes. Our results report an average of 30 runs.

B. Results

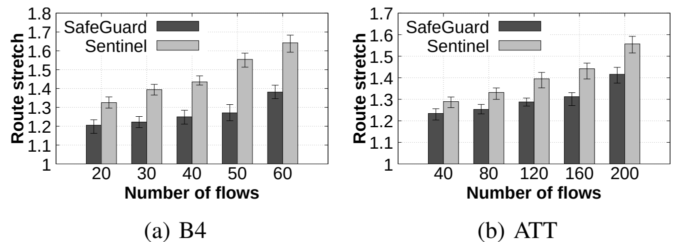
Link utilization. We start by discussing the impact of failures on the overall network utilization. Our interest here is to assess the susceptibility of SafeGuard and Sentinel to congestion after a link failure. For these experiments, we fix the number of flows at 60 for B4 and 200 for ATT, and randomly fail a link to measure the utilization of the remaining (active) links. Fig. 4 shows the CDF of link utilization across

[7]At the moment of writing this paper, Open vSwitch did not support weighted round-robin policies for traffic splitting.



(a) B4 (b) ATT

Fig. 4: FOR TEST CDF of link utilization for B4 and ATT topologies.



(a) B4 (b) ATT

Fig. 5: Average route stretch of SafeGuard and Sentinel after facing link failures.

all active links for both topologies after a failure. As can be seen, SafeGuard distributes the traffic load more uniformly than Sentinel. For B4 (Fig. 4a), neither approach led to congested links (i.e., utilization equal to one). However, 28% of the links are close to congestion with utilization higher than 80% in Sentinel. That number drops to 20% in SafeGuard.

For ATT (Fig. 4b), because of the higher number of flows, we also have higher link utilization compared to B4. Nevertheless, SafeGuard outperforms Sentinel with a bigger margin. For example, the load imposed by SafeGuard is above 80% for less than half of the links (around 40%) while in Sentinel, this number increases to 57%. That is because SafeGuard can use any active link to reroute traffic, even along the affected primary route, while Sentinel requires link-disjoint paths and thus has fewer options to balance traffic. Note that both approaches result in some level of congestion after a failure. However, the number of congested links is 50% smaller in SafeGuard compared to Sentinel.

Route stretch. Next, we look at the length (number of hops) of the backup routes selected by SafeGuard and Sentinel when a failure occurs. Longer backup routes can degrade application performance due to their increased latencies. In these experiments, we generate traffic between different numbers of randomly selected source-destination pairs and calculate the *route stretch* [17] (i.e., the ratio between the length of the longest path used by flow and the length of the shortest possible path for that flow) for both approaches after failing a link. Fig. 5a shows the average route stretch for the B4 topology. We can see that SafeGuard outperforms Sentinel for all amounts of flows. In particular, it can lead to backup routes up to 15% smaller than Sentinel for 60 flows (route stretch of 1.38 against 1.64). We observe a similar trend for the ATT topology (Fig. 5b), where SafeGuard performs better for a higher number of flows. For example, it results in backup routes around 10% smaller than Sentinel for 200 flows.

VI. Related work

Failure recovery in SDN. In addition to Sentinel [7], already discussed throughout the paper, there has been a large body of work on developing failure recovery strategies for software-defined networks [11], [18]–[23]. In common, most of them require customized headers (i.e., extensions to the OpenFlow protocol) and/or network device modifications to be deployed. Revive [24] proposes to systematically construct routing topology based on [25] to proactively install alternative routes on a subset of switches between a given source-designation pair. SD-FAST [26] extends this idea and ensures that fail-free traffic does not traverse tables associated with failure-recovery operations in the packet processing pipeline. Plinko [27] and Zhu *et al.* [28] propose compressing flow tables to reduce switch memory demands while deploying backup routes in SDN switches. These efforts are complementary to SafeGuard and can be used to reduce its memory footprint.

WAN traffic engineering. Many prior works focus on traffic engineering solutions for efficient failure recovery in wide-area networks. *CASA* [29] defines multiple link-disjoint arborescences (directed graphs with a single route from any node to the root) for each source-destination pair. Forward fault correction (FFC) [6] sets a bandwidth cap to support up to k arbitrary failures. TeaVaR [30] considers the failure probability of each link to maximize the network utilization while maintaining an expected availability. SMORE [31] constructs semi-oblivious routes between communicating pairs and distributes traffic among these routes while minimizing the maximum link utilization. Unlike SafeGuard, none of these approaches target SD-WANs and their tight switch memory capacities.

VII. Conclusion

Various traffic engineering mechanisms have been developed to ensure that software-defined WANs do not experience significant congestion in the presence of link failures. However, none of these mechanisms take practical switch memory constraints into account while rerouting impacted flows. In this paper, we propose a novel TE system for SD-WANs, called SafeGuard, that can quickly recover from failures by proactively allocating backup routes at the same time it accommodates both bandwidth and switch memory demands. We implemented a prototype of SafeGuard on top of the Ryu SDN controller and extensively evaluated it over two real topologies (Google B4 and ATT) using Mininet. Our results show that SafeGuard outperforms the state-of-the-art failure recovery scheme for SD-WANs, Sentinel.

Acknowledgements

We would like to thank the anonymous CNSM reviewers for their constructive feedback. An NSERC Discovery Grant partially supported this work.

References

[1] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 1–12, Aug. 2007.

[2] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, and S. Shenker, "Rethinking enterprise network control," *IEEE/ACM Transactions on Networking (TON)*, vol. 17, no. 4, pp. 1270–1283, August 2009.

[3] F. da Rocha, C. Paulo, and E. S. Mota, "A survey on fault managementin software-defined networks," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 4, pp. 2284–2321, August 2017.

[4] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, and C. Diot, "Characterization of failures in an IP backbone," in *IEEE International Conference on Computer Communications (INFOCOM)*. IEEE, 2004.

[5] N. L. Van Adrichem, B. J. Van Asten, and F. A. Kuipers, "Fast recovery in software-defined networks," in *Software Defined Networks (EWSDN), 2014 Third European Workshop on*. IEEE, 2014, pp. 61–66.

[6] H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter, "Traffic engineering with forward fault correction," in *SIGCOMM 2014*. ACM - Association for Computing Machinery, August 2014.

[7] J. Zheng, H. Xu, X. Zhu, G. Chen, and Y. Geng, "Sentinel: Failure recovery in centralized traffic engineering," *IEEE/ACM Trans. Netw.*, vol. 27, no. 5, pp. 1859–1872, 2019.

[8] B. Zhao, J. Zhao, X. Wang, and T. Wolf, "Ruletailor: Optimizing flow table updates in openflow switches with rule transformations," *IEEE Transactions on Network and Service Management*, pp. 1581–1594, 2019.

[9] "Safeguard," https://github.com/Meysam-Sh/SafeGuard.git.

[10] "Mininet," http://mininet.org.

[11] A. Capone, C. Cascone, A. Q. Nguyen, and B. Sanso, "Detour planning for fast and reliable failure recovery in sdn with openstate," in *Design of Reliable Communication Networks (DRCN), 2015 11th International Conference on the*. IEEE, 2015, pp. 25–32.

[12] j. D. S. Hartmanis, M. R., "Computers and intractability: a guide to the theory of np-completeness," *Siam Review*, 1982.

[13] Open Networking Foundation, "Openflow switch specification," September 2012.

[14] S. Jain, M. S. Kumar, A., J. Ong, L. Poutievski, A. Singh, and J. Zolla, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM Computer Communication Review*, pp. 3–14, 2013.

[15] "Att topology," http://www.topology-zoo.org/maps/AttMpls.jpg.

[16] "CpqD switch," git@github.com:CPqD/ofsoftswitch13.git.

[17] L. J. Cowen, "Compact routing with minimum stretch," in *Journal of Algorithms 38.1*. IEEE, 2001, pp. 170–183.

[18] D. Merling, W. Braun, and M. Menth, "Efficient data plane protection for sdn," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, 2018, pp. 10–18.

[19] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takacs, and P. Sköldström, "Scalable fault management for openflow," in *2012 IEEE International Conference on Communications (ICC)*, 2012, pp. 6606–6610.

[20] R. M. Ramos, M. Martinello, and C. Esteve Rothenberg, "Slickflow: Resilient source routing in data center networks unlocked by openflow," in *38th Annual IEEE Conference on Local Computer Networks*, 2013, pp. 606–613.

[21] M. Borokhovich, L. Schiff, and S. Schmid, "Provable data plane connectivity with local fast failover: Introducing openflow graph algorithms," ser. HotSDN '14, 2014, p. 121–126.

[22] W. Braun and M. Menth, "Loop-free alternates with loop detection for fast reroute in software-defined carrier and data center networks," *Journal of Network and Systems Management*, vol. 24, pp. 470–490, 2016.

[23] C. Cascone, L. Pollini, D. Sanvito, A. Capone, and B. Sansó, "Spider: Fault resilient sdn pipeline with recovery delay guarantees," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, 2016, pp. 296–302.

[24] I. Haque and M. Moyeen, "Revive: A reliable software defined data plane failure recovery scheme," in *2018 14th International Conference on Network and Service Management (CNSM)*. IEEE, 2018, pp. 268–274.

[25] I. Haque, S. Islam, and J. Harms, "On selecting a reliable topology in wireless sensor networks," in *Proceedings of the 2015 IEEE International Conference on Communications*, ser. ICC '15, 2015.

[26] M. Moyeen, , F. Tang, D. Saha, and I. Haque, "Sd-fast: A packet rerouting architecture in sdn," in *2019 15th International Conference on Network and Service Management (CNSM)*. IEEE, 2019.

[27] B. Stephens, A. L. Cox, and S. Rixner, "Scalable multi-failure fast failover via forwarding table compression," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '16, 2016.

[28] Z. Zhu, Q. Li, M. Xu, Z. Song, and S. Xia, "A customized and cost-efficient backup scheme in software-defined networks," in *2017 IEEE 25th International Conference on Network Protocol (ICNP)*. IEEE, 2017, pp. 1–6.

[29] K. Foerster, Y. Pignolet, S. Schmid, and G. Tredan, "Casa: Congestion and stretch aware static fast rerouting," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 469–477.

[30] R. B. . G. R. Supittayapornpong, S., "Striking the right utilization-availability balance in wan traffic engineering," *In Proceedings of the ACM Special Interest Group on Data Communication*, pp. 29–43, 2019.

[31] P. Kumar, Y. Yuan, C. Yu, N. Foster, R. Kleinberg, P. Lapukhov, C. L. Lim, and R. Soulé, "Semi-oblivious traffic engineering: The road not taken," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, Apr. 2018, pp. 157–170. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/kumar