We've Got You Covered: Failure Recovery with Backup Tunnels in Traffic Engineering

Jiaqi Zheng*†, Hong Xu*, Xiaojun Zhu‡, Guihai Chen†§, Yanhui Geng¶
*NetX Lab, City University of Hong Kong
†Nanjing University
‡Nanjing University of Aeronautics and Astronautics
§Shanghai Jiao Tong University
¶Huawei Noah's Ark Lab

***Abstract*—We present Sentinel, a novel failure recovery system for traffic engineering that pre-computes and installs backup tunnels to improve the robustness of software defined wide area networks (WANs). When a link fails, switches locally redirect traffic to backup tunnels and recover immediately in the data plane, thus substantially reducing the transient congestion compared to reactive rescaling. On the other hand Sentinel completely avoids the bandwidth headroom required by existing proactive approaches like FFC, and improves efficiency of operating the expensive WAN.**

We make several technical contributions in designing Sentinel. We formulate traffic engineering with backup tunnels (TEBT) as optimization programs. We propose an approximation algorithm to efficiently solve the problem. We further present a concrete design and implementation of the system based on Openflow group tables for backup tunnels. Extensive experiments on Mininet and numerical simulations show that similar to FFC, Sentinel reduces congestion by 45% compared with rescaling, and its algorithm runs much faster than FFC. Sentinel only introduces a small number of additional forwarding rules and can be readily implemented on today's Openflow switches.

I. Introduction

Centralized traffic engineering (TE) is widely used in practice to improve performance of wide area networks (WANs). Increasingly TE is implemented using software defined networking (SDN), where a logically centralized controller maintains a global view of the network state and dispatches TE plans as forwarding rules to the data plane. Usually tunnel based forwarding is used: the controller establishes multiple tunnels (i.e. network paths) between an ingress-egress switch pair, and configures splitting weights at the ingress switch for its traffic [10], [11], [15].

While a vast literature on TE exists, the issue of robustness is largely ignored. Failures are rather common in production networks with many network devices. Table I shows failure statistics data from Microsoft's data center WAN that connects its data centers worldwide [4]. The probability of having at least one link failure within five minutes, which corresponds to the TE frequency [10], [11], is more than 20%. Even with a single link failure, the impact can be severe as a SDN based WAN operates near capacity for efficiency [10], [11]. Say a 20 Gbps link fails and traffic is re-directed to another 100 Gbps link. Even with 100 MB buffer, the switch is unable to buffer after 40 ms, leading to a burst of packet loss.

TABLE I
Link failure frequencies in Microsoft data center WAN [4].

Number of link failure	Time intervals		
	2 min	5 min	10 min
1	10.6%	21.5%	31.2%
2	0.14%	1.1%	4.2%
3	0.14%	0.7%	1.4%

In general there are two approaches to failure recovery in TE: reactive and proactive, depending on whether failure is handled before happening or after. In data plane, the *de facto* reactive method is rescaling. Upon detecting the failure, the ingress switch normalizes splitting weights to re-direct traffic among the remaining tunnels [15]. Rescaling quickly restores connectivity. However it is purely local and may leave the network in a congested state. A better reactive approach is to resort to control plane intervention: the switch informs the controller, who can then compute a TE plan based on the new topology and update the entire network. This process takes substantially more time: according to [15], updating 100 rules on a single Openflow switch takes 1 second in median and 20 seconds in the worst case; it is also too slow to react to frequent failures in production settings.

To overcome these limitations, Liu et al. [15] propose a proactive approach called Forward Fault Correction (FFC). The idea is to proactively consider failures when formulating the TE problem, so that the TE solution can guarantee no congestion happens as long as the number of failures is at most k. FFC outperforms rescaling in performance and control plane intervention in responsiveness. Yet, it still suffers from several drawbacks. First, a portion of the network capacity has to be left vacant to guarantee that the network is congestion-free under arbitrary k failures. We find that, even with the lowest level of protection against just single link failures, FFC needs 100 Gbps–200 Gbps bandwidth headroom in Google's WAN topology with 38 100 Gbps links (details in Sec. VI). Though the overhead is only around 5%, leaving 200 Gbps capacity unused most of the time when no link is down translates to enormous amounts of monetary loss for the operator. Second, FFC needs to solve an LP with a large number of constraints and variables, which may be too slow for production networks.

We present Sentinel, a novel TE failure recovery system

using backup tunnels that has the benefits of both reactive and proactive approaches. In Sentinel, upon a link failure, the failing switch immediately activates the corresponding backup tunnels connecting itself to the egress switches of the flows affected by the failure, so the victim traffic on the failed tunnels can still reach their destinations. Moreover, Sentinel can also adjust the splitting weights at the original ingress switches for the victim traffic to reduce the congestion on the backup tunnels. Backup tunnels and splitting weights are pre-computed with global network state and installed at each switch in each TE interval. This enables fast and efficient failure recovery just like the proactive approach. On the other hand Sentinel does not need any bandwidth headroom; the network still runs at near perfect utilization without failure, preserving the utilization benefit of reactive approach.

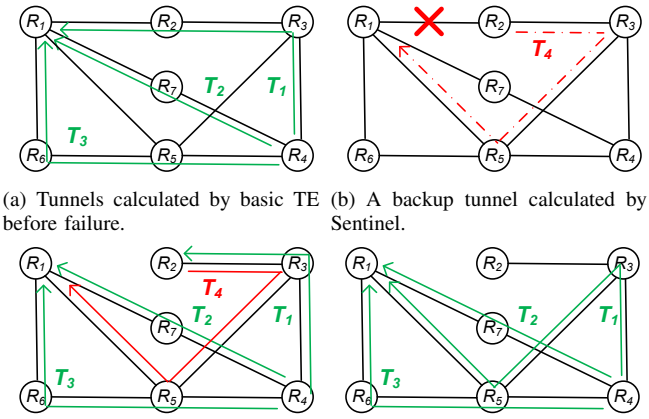
Inevitably, we cannot guarantee our approach is congestion-free with Sentinel. As argued above, since the network is fault-free most of the time, the bandwidth wastage required by congestion-free recovery methods far outweighs the congestion-induced traffic loss after failures, which eventually is taken care of by TCP in a time scale of seconds in the worst case. Thus it is only practical to use schemes like FFC just for high-priority traffic to reduce the bandwidth wastage [15]. Then most of the elastic traffic may still suffer from congestion, just as in Sentinel, when failures happen.

An immediate challenge of using backup tunnels is that there are exponentially many failure possibilities, and it is infeasible to account for all of them. In the current design Sentinel only considers single link failures, which in fact represents the overwhelming majority of failure scenarios. From Table I, for TE intervals of 2 or 5 minutes, multiple link failures happen rather infrequently. To reduce complexity and preserve scarce rule space on switches [8], leaving multiple link failures to controller intervention is a reasonable tradeoff.

We make three novel contributions in designing Sentinel. First, we propose a simple yet general optimization framework to model TE with backup tunnels (TE-BT) for single link failures. Compared to existing work that only considers routing over the remaining tunnels for failover, we additionally consider routing over the backup tunnels from the failing switch to egress switches. The two aspects are clearly coupled for the demand on backup tunnels is determined by routing at the flow's ingress switch. The objective is to minimize the maximum link utilization to reduce the transient congestion as much as possible. We formulate the TE-BT problem as a mixed integer program, which is hard to solve especially when the scale is large. We prove that this problem is NP-hard and $\Omega(\log \log n)$-hard to approximate where n is the number of switches.

Our second contribution is an efficient approximation algorithm to solve TE-BT in Sentinel. We relax the problem to an LP, which is still time-consuming to solve using standard solvers since we must solve it for every possible single link failure case at each TE interval. Inspired by the framework in [9], [14], we develop a fast approximation algorithm with a configurable accuracy parameter ϵ to solve this LP instead. Based on the fractional solution, we then apply randomized rounding to obtain a feasible solution of TE-BT. The complete algorithm yields a $(1+\epsilon)\mathcal{O}(\log n)$ upper bound for the maximum link utilization, where n is the number of switches.

Our third contribution is a concrete implementation and evaluation of Sentinel. We develop a prototype of Sentinel based on Openflow v1.3 using its fast failover group tables [3]. We evaluate the Sentinel prototype using Mininet with Google's data center WAN topology. We also conduct extensive simulations using realistic topologies to evaluate our algorithms in large-scale. Similar to FFC, Sentinel reduces congestion by 45% compared with rescaling, while completely avoiding the bandwidth headroom overhead of FFC. Sentinel only requires a small number of additional rules for backup tunnels, and its approximation algorithm runs much faster than FFC and standard LP methods.



(a) Tunnels calculated by basic TE before failure.

(b) A backup tunnel calculated by Sentinel.

(c) Active tunnels after the failure occurs.

(d) New tunnels re-calculated by basic TE after the failure.

Fig. 1. A simple motivation example.

TABLE II
DESCRIPTION OF TUNNELS USED IN FIG. 1.

Tunnel Id	Tunnel Type	Tag	Tunnel Description
1	primary	T_1	$\langle R_4, R_3, R_2, R_1 \rangle$
2	primary	T_2	$\langle R_4, R_7, R_1 \rangle$
3	primary	T_3	$\langle R_4, R_5, R_6, R_1 \rangle$
4	backup	T_4	$\langle R_2, R_3, R_5, R_1 \rangle$

II. A MOTIVATING EXAMPLE

Let us start with a toy example to illustrate the potential benefits of Sentinel. Consider the network in Fig. 1, in which there are seven switches $R_1, \ldots, R_7$, and all links have a capacity of 10 Gbps. We consider the flow from the ingress switch R_4 to egress switch R_1. Details of all the TE tunnels between this switch pair are shown in Table II. In Fig. 1(a), three tunnels T_1, T_2 and T_3 are established from R_4 to R_1. Our flow with 30 Gbps demand is routed with splitting weights of $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$, i.e. each tunnel carries 10 Gbps.

Now suppose the link from R_2 to R_1 fails, and tunnel T_1 fails immediately as well. In current SDN this is handled by data plane rescaling [15]. The ingress switch R_4 normalizes the splitting weights for the flow, which yields

$(0, \frac{1/3}{1/3+1/3}, \frac{1/3}{1/3+1/3}) = (0, \frac{1}{2}, \frac{1}{2})$, and rescales the flow onto the remaining tunnels T_2 and T_3. The load in these tunnels is $30 \times \frac{1}{2} = 15$ Gbps, which exceeds the link capacity by half. Thus local rescaling results in severe congestion and packet loss. Further, note that before R_4 detects the tunnel failure and applies rescaling, traffic is still routed on path $\langle R_4, R_3, R_2 \rangle$. These packets are simply dropped at R_2, wasting precious WAN bandwidth and creating unnecessary packet loss.

The state of the art, FFC, aims to guarantee that no congestion occurs after rescaling for k arbitrary failures. Assume the simplest case with $k = 1$. Then FFC cannot satisfy the flow demand requirement of 30 Gbps from R_4 to R_1 in the first place. The maximum flow between these two switches is only 20 Gbps in FFC, so that rescaling does not result in congestion in T_2 and T_3. Thus FFC results in 10 Gbps capacity loss. Moreover FFC does not re-direct the victim flow at R_2 affected by the failure neither.

Our idea is to use pre-installed backup tunnels to route the victim flow to the egress switch. Different from existing work that no longer use the failed tunnels, we rely on backup tunnels to recover them as much as possible to reduce congestion and improve performance. In this example, we can use T_4 as the backup tunnel as shown in Fig. 1(b). When packets from T_1 reaches R_2, R_2 applies the backup tunnel and routes them to R_3 and eventually R_1. This way R_2 does not have to drop any packets, and the flow can still reach its destination with the original rate of 10 Gbps without causing any congestion as shown in Fig. 1(c).

One may notice that the victim flow is rerouted at R_2 and causes redundant traffic from R_2 to R_3. We argue that this is acceptable since the main purpose of Sentinel is to quickly reroute traffic and minimize packet loss in the data plane when a link fails, instead of deploying the optimal TE solution to recover from the failure. Sentinel is the first line of defense and relies on simple rerouting as the temporary solution, before the controller updates the entire network based on the new optimal TE solution calculated from the new topology as shown in Fig. 1(d).

III. Sentinel Overview

On a high level, Sentinel is a TE failure recovery system that does the following at each TE interval: (1) computes and installs backup tunnels and splitting weights for all single link failures, and (2) activates them when a failure does happen.

We use our motivating example to illustrate how Sentinel works. At each interval, a baseline TE solution is first computed without any failure and configured by the controller as usual (i.e. T_1, T_2, T_3 each with a weight of 1/3). Based on the baseline TE solution, Sentinel then enumerates each link failure case, and computes backup tunnels to recover the failed tunnels, as well as new splitting weights at ingress switches to reduce congestion on backup tunnels for the failure.

The backup tunnels are configured using Openflow fast failover group tables designed specifically to detect and overcome port failures [3]. A group has a list of action buckets and each bucket has a watch port as a special parameter. The switch monitors liveness of the indicated port. If it is down, this bucket will not be used and the group quickly selects the next bucket (i.e. the backup tunnel) in the bucket list with a watch port that is up. In the running example, the backup tunnel is T_4, and the corresponding flow table and group table configurations for R_2 are shown in Table III. The backup splitting weights for each link failure case are configured using the group table and multi-level flow tables [3] as shown in Table IV for R_4. We have one flow table for each possible link failure case. These flow tables are sequentially numbered. The packets are first matched against flow entries of flow table 0 before going to the next flow table. Other flow tables may be used for different link failure cases.

Now using the running example, when the link fails, R_2 immediately detects the failure and applies the fast failover action in the group Gr 2.1. It changes the tunnel tag for the traffic from T_1 to T_4, and forwards packets destined to R_1 eastbound to R_3. At the same time, it sends a message to the controller about the link failure, and the controller instructs R_4 to modify the goto-table instruction in flow table 0 to match the packets to a different flow table (say flow table 2). Traffic is then routed according to a different group of

TABLE III
Flow table and group table at R_2 for the toy example.

Flow table at R_2

Match Field			Instruction
SrcPfx	DstPfx	Tag	
—	—	T_1	Gr 2.1

Group table at R_2

Group Identifier	Group Type	Action Buckets
Gr 2.1	Fast Failover	Outport: W Tag $\leftarrow T_4$; Outport: E

TABLE IV
Flow and group tables at R_4 in the toy example.

Flow table 0 at R_4

Match Field			Instruction
SrcPfx	DstPfx	Tag	
R_4	R_1	—	Goto Table 1

Flow table 1 at R_4

Match Field	Instruction
$*$	Gr 4.1

Flow table 2 at R_4

Match Field	Instruction
$*$	Gr 4.2

.....

Flow table i at R_4

Match Field	Instruction
$*$	Gr 4.i

Group table at R_4

Group Identifier	Group Type	Action Buckets
Gr 4.1	Select	Weight: $\frac{1}{3}$; Tag $\leftarrow T_1$; Outport: N Weight: $\frac{1}{3}$; Tag $\leftarrow T_2$; Outport: NW Weight: $\frac{1}{3}$; Tag $\leftarrow T_3$; Outport: W
Gr 4.2	Select	...
...	...	...
Gr 4.i	Select	...

action buckets, i.e. the new splitting weights for this failure over the original tunnels, as shown in Table IV. Note that the processing pipeline stops when a table does not include goto-table instruction. This completes the failover process in Sentinel.

We still rely on the controller in Sentinel. However the overhead of controller involvement is significantly decreased. The new splitting weights are calculated and pre-configured in the multi-level flow tables of ingress switches. In addition, we only modify the goto-table instruction (equivalent to a pointer) in the ingress switch to apply new weights, and thus update time can also be reduced. Note although we use multi-level flow tables, packets are actually processed with just two flow tables. The first one is table 0. The second one is table i, where the value of i depends on the failure case and is specified by goto-table instruction in flow table 0. Thus this does not incur much packets delay. We can also merge the items with the same splitting weights to save limited flow table space.

The central challenge in designing Sentinel is how to compute the backup tunnels and splitting weights with failures in TE, which is our focus in the following sections.

IV. An Optimization Framework

We introduce our optimization framework for traffic engineering with backup tunnels (TE-BT) in Sentinel. As discussed, we limit ourselves to considering single link failures for tractability, and leave multiple link failures which happen rarely to controller intervention.

A. Network Model

We first present our network model. A network is a directed graph $G = (V, E)$ where V is the set of switches and E the set of directed links. Each link l has a capacity C_l, and we use e to denote a particular failed link. The malfunctioning switch serving as the sending side of link e is denoted as s. Traffic can be carried over a set of pre-established tunnels (i.e. network paths). In SDN, flow tables are implemented using expensive TCAM with limited space for forwarding rules. Thus between each ingress-egress switch pair, we compute and establish a small number of edge-disjoint tunnels to preserve the scarce rule space. For example, 15 tunnels are available for any switch pair in Microsoft's SWAN [10]. We assume a baseline TE solution has been computed for the network without any failure, and the allocated rates of each flow on each tunnel are known.

When e fails, all tunnels traversing this link also fail immediately, but traffic continues to flow into these tunnels. Thus for each failed tunnel, we associate it with a *victim flow* $i, i = 1, 2, \ldots, m$, which needs to be routed from the failing switch s to its egress switch. We use p_i to denote the failed tunnel corresponding to i, and P_i the set of backup tunnels connecting s and i's egress switch without e. To avoid loops, we assume the backup tunnels do not involve i's original ingress switch. For example in Fig. 1(b), the victim flow is the traffic from R_2 destined to R_1 which is sent along T_1 to R_2, $p_i = T_1$, and $P_i = \{T_5\}$.

A victim flow i is always part of a unique *original flow* from its original ingress switch, and vice versa because all tunnels are edge-disjoint. For example in Fig. 1(b), the victim flow from R_2 to R_1 is part of the original flow of 30 Gbps from R_4 to R_1. We slightly abuse the notation and use the same index variable i to refer to the original flow corresponding to the victim flow i for convenience. The meaning of i is clear given the context as we will see soon. The demand of an original flow i, d_i, is the total bandwidth allocated to the flow in the baseline TE solution. This can only be changed by modifying the rate limiters at the hosts, and is much slower than switch failover. Thus we do not consider it as a degree of freedom in our problem. Finally, we use B_l to denote bandwidth used by traffic other than original and victim flows on link l.

It should be noted that a (victim or original) flow in our model is in fact an aggregate of all TCP flows between the same switch pair. This does not lose generality of the model, and is in line with previous work such as FFC [15]. For convenience, important notations are summarized in Table V.

TABLE V
KEY NOTATIONS IN THIS PAPER.

TE-BT Input	G	Network graph with switches S and directed links E
	$L_{p,l}$	1 if tunnel p uses link l and 0 otherwise
	e	The failed link
	m	Number of victim (original) flows
	P_i	Backup tunnel set for i
	p_i	Failed tunnel corresponding to i
	Q_i	Tunnels used for original flow i in the baseline TE
	d_i	Bandwidth demand of i from the baseline TE
	C_l	Capacity of link l
	B_l	Used bandwidth of link l
TE-BT Output	$x_{i,p}$	The allocated rate on tunnel $p \in P_i$ for victim flow i after e fails
	$y_{i,q}$	The allocated rate on tunnel $q \in Q_i$ for original flow i after e fails

B. TE-BT Formulation

We now present the TE-BT formulation. When link e fails, we consider two types of flows—victim flows and original flows—in order to recover from the failure. First, we need to obtain new routing for each victim flow i over the backup tunnels P_i, so that they can still reach their egress switches from the failing switch after the failure. This aspect has not been studied in previous work [15], [17]. Here we assume single path routing: each victim flow (i.e. failed tunnel) is rerouted via one backup tunnel to its egress switch. The main reasons are to save rule space in the switch flow tables and ease the implementation of failover which will be discussed in Sec. VI-A. Note it is possible to use multipath routing and we present the corresponding formulation in Sec. IV-C.

Second, we also need to compute the routing of original flows after the failure. Each original flow i is still routed through the same tunnels Q_i used in the baseline TE since the failed tunnel is recovered by a backup tunnel. Its splitting weights may change in order to reduce the traffic to the backup tunnel and congestion in the network given the demand.

The TE-BT problem is then to compute the optimal routing for both victim and original flows that minimizes the maximum link utilization λ in the network. Succinctly,

$$\min \quad \lambda, \tag{1}$$

$$\text{s.t.} \quad \sum_{i=1}^{m} \sum_{p \in P_i} x_{i,p} L_{p,l} + \sum_{i=1}^{m} \sum_{q \in Q_i} y_{i,q} L_{q,l} + B_l \leq \lambda C_l, \quad \forall l \in E \setminus e, \tag{1a}$$

$$\sum_{q \in Q_i} y_{i,q} = d_i, y_{i,q} \geq 0, \forall i, q, \tag{1b}$$

$$x_{i,p} = z_{i,p} y_{i,p_i}, z_{i,p} \in \{0,1\}, \sum_{p \in P_i} z_{i,p} = 1, \forall i, p. \tag{1c}$$

The LHS of (1a) characterizes the total link load, which takes both types of flows into account. Constraint (1b) is the usual flow conservation constraint with multipath routing for original flows. Constraint (1c) is the flow conservation constraint with single path routing for victim flows. The demand of victim flow i is equal to the rate of its original flow allocated to tunnel p_i, i.e. y_{i,p_i}, and $z_{i,p}$ is the binary variable indicating if i is routed to p or not.

Theorem 1: The TE-BT problem is not only NP-hard but also $\Omega(\log \log n)$-hard to approximate, where n is the number of switches.

Proof: We reduce the NP-hard congestion minimization problem [7] to the TE-BT problem. Without loss of generality, we consider a special case of TE-BT with only one tunnel for each source destination switch pair. In such cases, when link e fails, the victim flow is the original flow. For each tunnel routed through link e, we need to find a backup tunnel from the failing switch to its egress switch. Thus, the congestion minimization problem can be reduced to our problem and vice visa. Furthermore, since congestion minimization is $\Omega(\log \log n)$-hard to approximate unless NP $\subseteq$ DTIME $(n^{\mathcal{O}(\log \log \log n)})$ [7], the TE-BT problem is $\Omega(\log \log n)$-hard to approximate as well.

■

C. TE-BT with Multipath Routing for Victim Flows

If multipath routing is allowed over the backup tunnels for the victim flows, we only need to change the flow conservation constraint for victim flows in the formulation:

$$\min \quad \lambda, \tag{2}$$

$$\text{s.t.} \quad \text{(1a), (1b)} \tag{2a}$$

$$\sum_{p \in P_i} x_{i,p} = y_{i,p_i}, x_{i,p} \geq 0, \forall i, p. \tag{2b}$$

Clearly this is a classical multicommodity flow problem [5]. It can be solved in polynomial time using standard LP solvers, or even faster using fully polynomial time approximation algorithms [12]. Thus we focus on solving (1) in the following.

V. An Approximation Algorithm

The mixed integer program (1) can be relaxed to a linear program which is exactly the multipath routing formulation (2). The optimal fractional solutions $\{\tilde{x}_{i,p}\}$ and $\{\tilde{y}_{j,q}\}$ of the relaxed LP can be obtained in polynomial time using standard solvers. However, solving this LP is time-consuming especially when the number of flows is large, and we must solve the LP for every possible single link failure case at each TE interval (5 minutes usually) to increase robustness of data plane. Thus we set out to find an efficient approximation algorithm to solve the LP instead. Inspired by the framework in [9], [14], we develop a fast approximation algorithm shown in Algorithm 1. Generally, Algorithm 1 consists of finding initial routing, which may not be feasible, and continuously reroute flows until it is within $(1+\epsilon)$ of optimal solution, where ϵ is an accuracy parameter and as an input of our algorithm. We then apply randomized rounding [19] to obtain a feasible solution to (1). The entire algorithm is shown in Algorithm 3.

Algorithm 1 Fast approximation algorithm

Input: Network topology $G' = (V, E \setminus e)$; backup tunnel set P_i; tunnels set Q_j used in the baseline TE; accuracy ϵ.
Output: The approximate fractional $\{\tilde{x}_{i,p}\}$ and $\{\tilde{y}_{j,q}\}$ to (2).

1: **for** each i **do**
2: **for** each $q \in Q_i$ **do**
3: $\tilde{y}_{i,q} = d_i \cdot \frac{\phi(q)}{\sum_{q \in Q_i} \phi(q)}$, where $\phi(q) = \arg\min_{l \in q} C_l$.
4: **end for**
5: **for** each $p \in P_i$ **do**
6: $\tilde{x}_{i,p} = \tilde{y}_{i,p_i} \cdot \frac{\phi(p)}{\sum_{p \in P_i} \phi(p)}$
7: **end for**
8: **end for**
9: Calculate $\{\tilde{\lambda}_l\}$ corresponding to $\{\tilde{x}_{i,p}\}$ and $\{\tilde{y}_{j,q}\}$
10: $\tilde{\lambda} = \max_{l \in E} \tilde{\lambda}_l$
11: $\beta = 2(1+\epsilon)\ln(|E|\epsilon^{-1})\tilde{\lambda}^{-1}\epsilon^{-1}$
12: **repeat**
13: $\gamma = \frac{\epsilon}{8\beta\tilde{\lambda}}$
14: **for** each $l \in E$ **do**
15: $H_l = \frac{e^{\beta \cdot \tilde{\lambda}_l}}{C_l}$, where e is Euler's Number.
16: **end for**
17: Run Algorithm 2 to obtain solutions $\{\tilde{x}_{i,p}\}$ and $\{\tilde{y}_{j,q}\}$
18: Update $\{\tilde{\lambda}_l\}$ and $\tilde{\lambda}$ corresponding to $\{\tilde{x}_{i,p}\}$ and $\{\tilde{y}_{j,q}\}$
19: **until** (R1) and (R2) hold

We now explain the high level working of Algorithm 1. We first obtain an initial fractional solution by assigning rates to flows proportional to the bottleneck capacity of their available tunnels (lines 1-8). Then we calculate the load of each link $\tilde{\lambda}_l$, and the current cost of each link H_l as defined in line 15. Note that link cost increases exponentially with the link load. Now with this link cost concept, we iteratively adjust each flow's routing in order to gradually reduce the congestion (lines 12-19), until two relaxed optimality conditions are satisfied which will be discussed soon.

The rerouting procedure is shown in Algorithm 2. First we calculate M_i, the cost of current routing $\tilde{x}, \tilde{y}$ for each i based on link costs $\{H_l\}$, and sort them in a descending order of M_i (lines 1-2). Then we start from the flows with the highest routing cost (i.e. worst routing) and try to reroute them. We obtain the best routing $\{\hat{x}_{i,p}\}$ and $\{\hat{y}_{i,q}\}$ that minimizes the routing cost M_i without increasing $\tilde{\lambda}$ (line 5). This can be easily done by just routing the original and victim flows to

Algorithm 2 Reroute algorithm

Input: Initial solution $\{\tilde{x}_{i,p}\}$ and $\{\tilde{y}_{i,q}\}$; parameter γ.
Output: The solution $\{\tilde{x}_{i,p}\}$ and $\{\tilde{y}_{i,q}\}$ after rerouting.
1: Calculate the cost of routing M_i for each i, where $M_i = \sum_{p\in P_i} \tilde{x}_{i,p} L_{p,l} H_l + \sum_{q\in Q_i} \tilde{y}_{i,p} L_{q,l} H_l$.
2: Sort $\{i\}$ according to M_i in a descending order.
3: **for** each i **do**
4: **if** $i \in A$ **then**
5: Calculate $\{\hat{x}_{i,p}\}$ and $\{\hat{y}_{i,q}\}$ to minimize M_i.
6: **for** each $q \in Q_i$ **do**
7: $\tilde{y}_{i,q} = (1-\gamma)\tilde{y}_{i,q} + \gamma\hat{y}_{i,q}$
8: **end for**
9: **for** each $p \in P_i$ **do**
10: $\tilde{x}_{i,p} = (1-\gamma)\tilde{x}_{i,p} + \gamma\hat{x}_{i,p}$
11: **end for**
12: **break**
13: **end if**
14: **end for**

the tunnel with most inexpensive total cost first before the bottleneck link reaches $\tilde{\lambda}$, and switch to the second most lightly used tunnel and so on until the demands are satisfied. Because the link cost is an exponential function of its load that magnifies the impact of congestion, it punishes the rerouting from choosing highly congested links and tunnels, and the new routing $\{\hat{x}_{i,p}\}$ and $\{\hat{y}_{i,q}\}$ is guaranteed to reduce the maximum link congestion in the network [14]. We then obtain a new routing by rerouting γ portion of the flow according to the new routing and reserving $1-\gamma$ portion with the original routing (lines 5-12).

We now complete the algorithm by giving the two relaxed optimality conditions with the following theorem from [14]:

***Theorem** 2:* [14]. Let $\{\tilde{x}_{i,p}\}$ and $\{\tilde{y}_{i,p}\}$ be the solution to (2) computed by Algorithm 1, which satisfies constraint (1a). Then for a given $\epsilon > 0$, this solution is ϵ-optimal if the following relaxed optimality conditions (R1) and (R2) hold:

(R1): For each edge $l \in E$, either (3) or (4) holds:

$$\sum_i \sum_{p\in P_i} x_{i,p} L_{p,l} + \sum_i \sum_{q\in Q_i} y_{i,q} L_{q,l} + B_l \geq \frac{\tilde{\lambda}\cdot C_l}{(1+\epsilon)}, \quad (3)$$

$$C_l H_l \leq \frac{\epsilon}{|E|} \sum_{l'\in E} (C_{l'} H_{l'}), \quad (4)$$

where $\tilde{\lambda}$ and H_l are defined in Algorithm 1 (lines 12 and 17).

(R2): The following holds:

$$\begin{aligned} &\sum_{i'\in A} M_{i'} \leq \epsilon \sum_{i=1}^{m} M_i, \\ &A = \left\{ i \middle| M_i - \hat{M}_i > \epsilon M_i + \frac{\epsilon\tilde{\lambda}}{m} \sum_{l\in E} C_l H_l \right\}, \end{aligned} \quad (5)$$

where M_i is the cost of the routing $\{\tilde{x}_{i,p}\}$ and $\{\tilde{y}_{i,p}\}$ defined in line 1 of Algorithm 2, and $\hat{M}_i$ represents the minimum cost of routing just the victim and original flows i given $\{\tilde{\lambda}_l\}$.

Note the parameter β (line 11 in Algorithm 1) is selected so that relaxed optimality condition (R1) is always satisfied [14]. The process of rerouting flow gradually enforces the relaxed optimality condition (R2). When both (R1) and (R2) are satisfied, the procedure stops and we obtain ϵ-optimal fractional solutions $\{\tilde{x}_{i,p}\}$ and $\{\tilde{y}_{i,q}\}$.

Algorithm 3 TE-BT Solution Algorithm

Input: Network topology $G' = (V, E-e)$; backup tunnel set P_i; tunnels set Q_j used in the baseline TE.
Output: The solution $\{x_{i,p}\}$ and $\{y_{j,q}\}$ to (1)
1: Run Algorithm 1 and obtain solutions $\{\tilde{x}_{i,p}\}$ and $\{\tilde{y}_{j,q}\}$
2: **for** each i **do**
3: **for** each $q \in Q_i$ **do**
4: $y_{i,q} = \tilde{y}_{i,q}$
5: **end for**
6: $\hat{P} = \emptyset$
7: **for** each $p \in P_i$ and $p \notin \hat{P}$ **do**
8: $\hat{P} = \hat{P} \cup p$
9: $z_{i,p} = \frac{\tilde{x}_{i,p}}{y_{i,p_i}}$
10: $l_{i,p} = \sum_{p\in\hat{P}} z_{i,p}$
11: **end for**
12: Generate a number r in (0,1] uniformly at random
13: Find $\hat{p}$ such that $r \leq l_{i,p}$ and $l_{i,p} - r$ is minimum
14: $x_{i,\hat{p}} = y_{i,p_i}$
15: **end for**

Finally, Algorithm 3 shows the entire solution algorithm that transforms the fractional solution from Algorithm 1 to a feasible solution to the TE-BT problem by randomized rounding. For $\{\tilde{y}_{i,q}\}$, it is already a feasible solution (lines 3-5). For $\{\tilde{x}_{i,p}\}$, we apply randomized rounding to obtain an integer solution $\{x_{i,p}\}$ (lines 6-14). To ensure that only one tunnel is chosen for victim flow i, the fractional solution can be viewed as partitioning the interval $[0,1]$ to intervals of lengths $\{z_{i,p}\}$ (lines 8-10). A real number is generated uniformly at random in $(0,1]$ and the interval in which it lies determines the tunnel (lines 12-14).

Before analyzing the performance of Algorithm 3, we introduce some necessary notations. Let $Z_{i,p}$ be a binary random variable that indicates whether flow i is routed through tunnel $p \in P_i$. If i passes through tunnel p, $Z_{i,p} = 1$, otherwise $Z_{i,p} = 0$. Let $y_l = \sum_i \sum_{q\in Q_i} \tilde{y}_{i,q} L_{q,l}$, which represents the maximum load on link l for the original flow when link e fails, and let $Z_l = \sum_i \sum_{p\in P_i} Z_{i,p} \tilde{y}_{i,p_i} + y_l + B_l$ be a random variable that indicates the total maximum load on link l when link e fails.

Let $\tilde{\lambda}_{opt}$ be the optimum of (2), and λ_{opt} be the actual optimum of (1). Clearly $\tilde{\lambda}_{opt} \leq \lambda_{opt}$.

***Theorem** 3:* Algorithm 3 outputs a feasible solution bounded by $\mathcal{O}(\log n)(1+\epsilon)\lambda_{opt}$ with high probability, where n is the number of switches in the network.

Proof: Fractional solution $\{\tilde{x}_{i,p}\}$ and $\{\tilde{y}_{j,q}\}$ are obtained by Algorithm 1. Thus,

$$\tilde{\lambda} \leq (1+\epsilon)\tilde{\lambda}_{opt} \quad (6)$$

According to constraints (1a) and (6),

$$\begin{aligned}
E[Z_l] &= \sum_i \sum_{p\in P_i} E\left[Z_{i,p}\cdot \tilde{y}_{i,p_i}\right] + y_l + B_l \\
&= \sum_i \sum_{p\in P_i} \Pr\left[Z_{i,p}=1\right]\cdot \tilde{y}_{i,p_i} + y_l + B_l \\
&= \sum_i \sum_{p\in P_i} z_{i,p}\cdot \tilde{y}_{i,p_i} + y_l + B_l \\
&\le \tilde{\lambda} C_l \\
&\le (1+\epsilon)\tilde{\lambda}_{opt} C_l \\
&\le (1+\epsilon)\lambda_{opt} C_l
\end{aligned}$$

Let $W_l = \frac{Z_l}{C_l} = \sum_i \sum_{p\in P_i} Z_{i,p}\cdot \frac{\tilde{y}_{i,p_i}}{C_l} + \frac{y_l+B_l}{C_l}$. From above, $E[W_l] \le (1+\epsilon)\lambda_{opt}$. The random variables $\{Z_{i,p}\}$ are mutually independent since tunnel p for flow i is chosen independently in Algorithm 3. Therefore, W_l, the sum of random variables $\{Z_{i,p}\}$, is independent. Choose δ such that $(1+\delta) = \frac{8\ln n}{\ln\ln n}$ and apply Chernoff bound [19],

$$\Pr\left[W_l \ge \frac{8\ln n}{\ln\ln n}(1+\epsilon)\lambda_{opt}\right] \le \left(\frac{8\ln k}{e\ln\ln n}\right)^{-8\frac{\ln n}{\ln\ln n}} \le \frac{1}{n^4}.$$

There are n switches in the network, so the number of links between nodes is at most n^2. Let $\sigma = \frac{8\ln n}{\ln\ln n}(1+\epsilon)\lambda_{opt}$ and by union bound,

$$\Pr\left[\max_{l\in E} W_l \ge \sigma\right] \le \sum_{l\in E} \Pr\left[W_l \ge \sigma\right] \le n^2\cdot \Pr\left[W_l \ge \sigma\right] \le \frac{1}{n^2}.$$

■

VI. Experimental Evaluation

We evaluate Sentinel using both prototype implementation and large-scale simulation.

Benchmark Schemes: We compare the following schemes with Sentinel.

- **SR**: The baseline failover mechanism using simple rescaling in the data plane.
- **FFC**: State-of-the-art proactive failure recovery mechanism [15]. We configure FFC to handle arbitrary single link failure, a setting identical to Sentinel.

For Sentinel, unless stated otherwise, we configure the accuracy parameter ϵ to 1.0 in Algorithm 1.

A. Implementation and Mininet Emulations

Implementation: We develop a prototype of Sentinel using the Floodlight 1.1 [1] controller with Openflow v1.3. The forwarding rules are installed and updated via Floodlight's REST API. We use VLAN IDs as labels to perform tunnel-based forwarding. Ingress switches assign VLAN ID to each flow and intermediate switches simply forward packets based on the input port and VLAN ID.

At each TE interval, after obtaining the baseline TE solution, Sentinel does the following iteratively: A link is removed from the topology to represent link failure. For each tunnel on this link, we compute a corresponding backup tunnel set that connects the malfunctioning switch to the tunnel's egress switch, taking into account the current flow table space constraint. It then applies Algorithm 3 to solve the TE-BT problem (1), assigns new VLAN IDs to the computed backup tunnels, and installs them on their corresponding switches. It also installs the computed splitting weights in the multi-level flow tables of the tunnels' ingress switches. We use floodlight `setTableId()` function to specify the index for different flow tables. There are at most 256 flow tables in each switch, whose index ranges from 0 to 255. We use various standard Openflow messages in our implementation. Notably, when a link fails, the switch sends an `OFPT_PORT_STATUS` message about the port down event to the controller. The controller then sends an `OFPT_FLOW_MOD` message to modify the goto-table instruction using the pre-computed mapping from link failure case to goto-table instruction.

Note that in Sentinel, flow table configurations for ingress switches are different from intermediate switches and require special attention. Ingress switches carry out two functions: flow splitting and failover when its incident links fail. Both require the use of group tables. However, one group in an Openflow group table can only be configured to one type and perform one function. Thus we need to cascade multiple groups for ingress switches in our implementation. For example for R_1, the action of flow table 0 points to table 1, which is implemented by floodlight `buildGotoTable()` function. Flow table 1 points to the first group Gr 1.1 of type `select` as shown in Table VI. Its group table is shown in Table VII. Gr 1.1 performs multipath routing over three tunnels in the normal case. It also cascades a `fast failover` group for each tunnel for their backup tunnels, which is only effective when the watch port is down. Thus R_1's group table has four groups in total.

TABLE VI
An example of a flow table in an ingress switch.

Flow table 0 at R_1

Match Field			Instruction
SrcPfx	DstPfx	Tag	
R_1	R_{12}	—	Goto Table 1

Flow table 1 at R_1

Match Field	Instruction
*	Gr 1.1

For completeness we explain the implementation of rescaling and FFC now. Rescaling usually relies on some data plane link monitoring protocol to detect link failures and notify corresponding ingress switches [15]. We simplify it with a control plane implementation: The Floodlight controller monitors the link status using its APIs, and directly notifies ingress switches to rescale traffic upon a failure. We implement algorithms in [15] to compute TE solutions for FFC. When a failure happens FFC still uses rescaling.

Mininet Setup: We conduct experiments on Mininet 2.2.1 [13], a high fidelity network emulator for SDN, running on a PC with an Intel i5-2400 quad-core processor. We use OpenvSwitch version 2.3.1. Due to the single machine

TABLE VII
AN EXAMPLE OF A GROUP TABLE FOR THE INGRESS SWITCH R_1.

Group Identifier	Group Type	Action Buckets
Gr 1.1	Select	Weight: $\frac{1}{4}$; Tag $\leftarrow T_1$; Gr 1.2 Weight: $\frac{1}{4}$; Tag $\leftarrow T_2$; Gr 1.3 Weight: $\frac{1}{2}$; Tag $\leftarrow T_3$; Gr 1.4
Gr 1.2	Fast Failover	Outport: E Tag $\leftarrow T_4$; Outport: SE
Gr 1.3	Fast Failover	Outport: SE Tag $\leftarrow T_5$; Outport: S
Gr 1.4	Fast Failover	Outport: S Tag $\leftarrow T_6$; Outport: E

limitation of Mininet, we adopt a small scale WAN topology for Google's inter-data center network reported in [11]. There are 12 switches and 38 links as illustrated in Fig. 2. We set link bandwidth to 700 Mbps with 1 ms delay, and switch per-port buffer size 1 M. We use the `link down` command in Mininet to simulate failures.

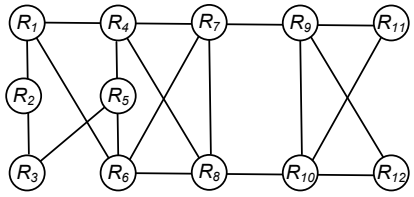


Fig. 2. The Google data center WAN topology used in Mininet.

Experiment Results: We evaluate the effectiveness of Sentinel and FFC in handling link failures. In this set of experiments, we generate 20 UDP flows in each run, and fail a link randomly in the topology. We focus on the victim flows affected by the failed link, and vary its rate from 0 Mbps to 500 Mbps at the increment of 10 Mbps. We perform the same experiment with 10 runs for both Sentinel and FFC, and report the average values of numbers of lost packets and out-of-order packets measured by `iperf` for the victim flows.

Fig. 3 depicts the number of lost packets comparison. We can see that, as the rate of the victim flows increases, SR and FFC result in dramatically more lost packets while Sentinel only leads to a mild number of lost packets. Specifically, when the flow rate is 500Mbps, Sentinel reduces lost packets by 87.8% and 70.8% compared with SR and FFC. Packets are lost right after the link failure because it takes the switch some time to perform failover. For Sentinel, the switch quickly applies the backup tunnel defined in its fast failover group to reroute the victim flows. This is done purely locally in data plane and leads to very few lost packets. In SR, as well as FFC, there is no backup tunnel. The victim flows are rerouted by rescaling which is activated at the ingress switch rather than the failing switch, which takes longer to do and results in more lost packets. SR has more lost packets than FFC as a result of the severe congestion after rescaling, which is beyond the 1Mb buffer in our experiments.

We now look at the number of out-of-order packets as shown in Fig. 4. When the flow rate is less than 300 Mbps, Sentinel and FFC perform almost the same. When the flow rate is larger, however, Sentinel has more severe reordering. We believe it is because the backup tunnels used in Sentinel typically lead to longer hop distance compared to the failed tunnel, whereas in FFC the victim flows, after rescaling, are routed via the remaining tunnels which have similar hop distances with the failed tunnel. The out-of-order packets of SR are significantly more than Sentinel and FFC especially when the rate of victim flows is larger. That is because a larger rate aggravates congestion and yields more severe reordering.

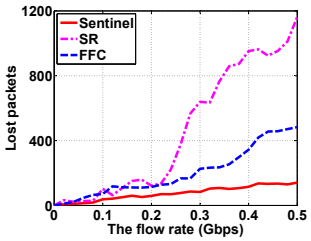


Fig. 3. Number of lost packets comparison.

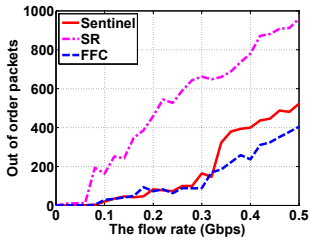


Fig. 4. Number of out-of-order packets comparison.

In the second experiment, we measure packet loss rate (PLR) for UDP flows after link failures. Packet loss rate is the ratio between the total number of lost packets and the total number of sent packets from ingress to egress switch. Table VIII shows the average results with at least 20 runs when the link $\langle R_5, R_6 \rangle$ fails. We generate 600 3.5Mbps flows for Sentinel and SR to maximize network utilization. For FFC we can only generate at most 400 3.5Mbps flows, beyond which congestion-free is not always guaranteed with the failure. Each run of the experiment lasts for 60 seconds. When all the flows start, we use the `link down` command in Mininet to simulate link failure, and record packet drops in different schemes. We observe that the average packet loss rate of Sentinel is around 0.1%; that is, the impact is negligible to most applications. Sentinel adapts to network dynamics well by quickly applying the correct set of forwarding rules defined in the group tables. The average packet loss rates of SR and FFC are 1.118% and 0.202%, which are larger than Sentinel. This is because the number of congested links using SR is larger than that using Sentinel when the link fails. In addition, SR and FFC simply use rescaling at the ingress switches, which causes packet drops for the victim flows and further degrade its performance. Compared with FFC, Sentinel is able to offer better performance without any vacant capacity reserved at links and zero throughput loss. Finally we note that the reason of a small extent of packet drops in Sentinel is that the hashing based flow splitting in OpenvSwitch is imperfect due to the probabilistic nature.

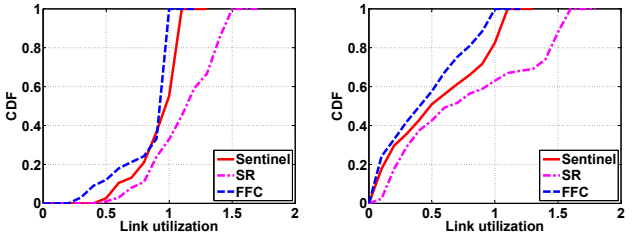
B. Simulation

We also conduct extensive simulations to thoroughly evaluate Sentinel at scale.

Setup. In addition to the small-scale Google topology used in Mininet experiments, here we use a large-scale synthetic scale-free topology that is randomly produced by the `scale_free_graph` function in [2], referred to as ScaleFree topology. There are 100 switches and 586 100 Gbps links

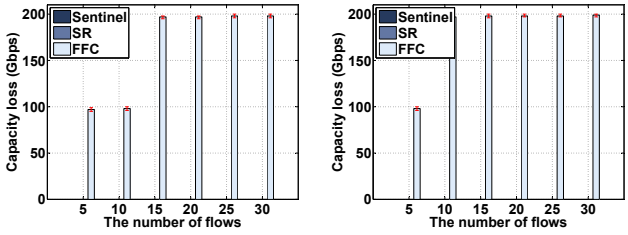
TABLE VIII
PERFORMANCE IN DIFFERENT SCHEMES WHEN ONE LINK FAILS.

	Type	Volume of each flow	Number of flows	Throughput loss	Average PLR
Sentinel	UDP	3.5Mbps · 60s	600	0	0.131%
SR	UDP	3.5Mbps · 60s	600	0	1.118%
FFC	UDP	3.5Mbps · 60s	400	700M	0.203%



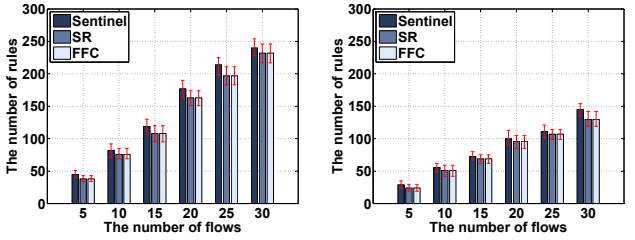
(a) Google topology (b) ScaleFree topology

Fig. 5. The link utilization comparison.



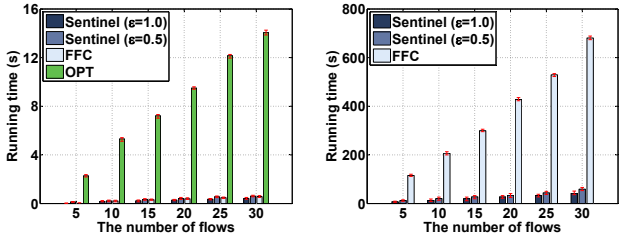
(a) Google topology (b) ScaleFree topology

Fig. 6. The capacity loss comparison.



(a) Google topology (b) ScaleFree topology

Fig. 7. The comparison of the maximum number of rules in a switch.



(a) Google topology (b) ScaleFree topology

Fig. 8. Running time comparison.

in total. For each topology, we generate different numbers of flows for unique ingress-egress switch pairs. Recall in our model a flow is an aggregate of traffic between a unique switch pair as discussed in Sec. IV-A. A flow's demand is fixed at 5 Gbps in the Google topology, and 15 Gbps in the ScaleFree topology. We use at most 15 tunnels between any switch pair. We run the algorithms on an Amazon EC2 `c4.2xlarge` instance with 8 CPUs and 15 GB memory. Each data point is an average of three runs.

Failover Performance. We first investigate the failure recovery performance of different schemes, by comparing the average link utilization of the network after a link failure. Our simulation does not emulate the flow-level behavior and thus we cannot measure the lost packets during the transient stage as in the Mininet experiments. Intuitively, congestion happens when the utilization is larger than one, and a larger value indicates more severe congestion in the network. Fig. 5 shows the CDFs of link utilization across active links that carry traffic for different schemes. For this simulation we fix the number of flows at 15 for the Google topology, and 30 for ScaleFree. FFC guarantees the network congestion-free even with failures, and we can see its link utilization is always less than or equal to one for both topologies. Sentinel cannot make such guarantees since it does not leave any bandwidth headroom when the network is operating normally. Surprisingly, we find that Sentinel performs very close to FFC with just slightly more congestion especially in the Google topology in Fig. 5(a). The maximum link utilization is ∼1.1. Rescaling, i.e. SR in the figures, results in severe congestion. This demonstrates that Sentinel in general leads to a small degree of congestion compared to state of the art approaches, and significantly outperforms rescaling by around 45%.

As discussed FFC requires certain portion of network capacity to be left vacant to make congestion-free guarantees. To quantify this overhead, Fig. 6 plots the capacity loss, defined as the difference between the theoretical maximum throughput achievable given the flows' ingress and egress switches, and the actual throughput achievable in FFC. Notice that this is the throughput when the network has no failure. We can observe that the maximum capacity loss for FFC is about 200 Gbps when the number of flows is larger than 15 and 10 in Google and ScaleFree topology, respectively. Both SR and Sentinel clearly have no capacity loss as they do not require any change to the TE without failure. Thus, Sentinel substantially reduces the bandwidth overhead of FFC while providing similar failure recovery performance.

Rule Space Overhead. We now look at the rule space overhead of Sentinel with backup tunnels. If one tunnel travels through a switch, it occupies one physical rule in the flow table. If an additional backup tunnel (in the group table) is added by Sentinel, it occupies two physical rules. Fig. 7 shows the maximum number of rules required in a switch for FFC, SR and Sentinel in Google and ScaleFree topology. SR and FFC do not use backup tunnels and the maximum number of

tunnels for them is the same. Sentinel needs additional backup tunnels to reroute victim traffic and reduce congestion, and it requires 10% more rules compared with SR and FFC.

Algorithm Running Time. Finally we evaluate the running time of our algorithm which is illustrated in Fig. 8. Besides FFC, we compare our algorithm against a branch and bound method that solves the TE-BT problem (1) optimally, denoted as OPT. For Sentinel, we set ϵ to be 0.5 and 1.0 respectively. We can observe that in the Google topology, the running time of Sentinel and FFC are both less than 1 second for up to 30 flows, while OPT takes more than 12 seconds when the number of flows is beyond 25. In the ScaleFree topology, OPT does not complete even after two hours and is orders of magnitude longer than other schemes. Thus we do not include it in Fig. 8(b). Sentinel's running time is less than 60 seconds even when the number of flows is 30, while FFC takes more than 600 seconds. Moreover, we observe an intuitive tradeoff between solution accuracy and running time for Sentinel's approximation algorithm: with more accurate approximation, i.e. a smaller ϵ, it takes longer to compute the solution. This can be used in practice to further speed up the algorithm.

VII. Related Work

We briefly review prior art on failure recovery in SDN. Relying on Openflow local fast failover mechanisms, Liu et al. [16] propose new routing approaches, and Borokhovich et al. [6] develop candidate path search algorithms to guarantee connectivity in the data plane. Sentinel not only ensures connectivity but also strives to minimize the failure induced congestion. Recent work also aims to reduce congestion during failures. Suchara et al. [17] consider pre-computing the ingress switch splitting weights for arbitrary k faults to prevent rescaling induced congestion. This approach does not work for large scale production networks due to the exponentially many failure cases. R3 [18] proposes using both proactive offline planning and a reactive fast re-route protocol, and FFC [15] develops a purely proactive approach to provide congestion-free guarantees for a large number of failure scenarios. As discussed extensively in Sec. I, this line of work introduces substantial bandwidth overhead that outweighs the potential benefit of eliminating transient congestion, while Sentinel does not suffer from this drawback. Finally, though the idea of using backup tunnels and pre-computation has surfaced (sometimes implicitly) in the literature, our use of backup tunnels is different. In current schemes, traffic is still routed to the failed tunnels which results in packet drops and bandwidth wastage before the failover is completed at the ingress switch. We use backup tunnels that start from the failing switch and end at the egress switches to re-direct the affected traffic, in addition to adjusting weights, and is therefore faster and more effective in reducing transient congestion. Our novelty also lies in a comprehensive exploration of using backup tunnels in a software defined WAN, which to our knowledge has not been done before.

VIII. Conclusion

In this paper, we presented Sentinel, a novel failure recovery system that uses backup tunnels to redirect victim traffic. We developed efficient approximation TE algorithms with fast runtime to compute the backup tunnels. We also implemented a prototype of Sentinel based on Openflow v1.3 using its fast failover group tables. Experiments on Mininet and numerical simulations with real-world WAN topologies demonstrate that Sentinel can effectively reduce traffic loss and increase capacity utilization.

Acknowledgment

We thank Ming Zhang, Harry Liu, Jean Walrand, and the anonymous reviewers for helpful comments. This work was supported by Huawei under grant no. 9231208 and was partly supported by China 973 projects (2014CB340303), and China NSF grants (61672353, 61472252, 61321491, 61133006, 61502232).

References

[1] Floodlight. http://floodlight.openflowhub.org/.
[2] Networkx. https://networkx.github.io/.
[3] Openflow switch specification. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf.
[4] Private conversation with researchers at Microsoft Research, March 2015.
[5] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.
[6] M. Borokhovich, L. Schiff, and S. Schmid. Provable data plane connectivity with local fast failover: Introducing openflow graph algorithms. In *HotSDN*, pages 121–126. ACM, 2014.
[7] J. Chuzhoy and J. Naor. New hardness results for congestion minimization and machine scheduling. *Journal of the ACM*, 53(5):707–721, 2006.
[8] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: scaling flow management for high-performance networks. In *SIGCOMM*, pages 254–265, 2011.
[9] L. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM J. Discrete Math.*, 13(4):505–520, 2000.
[10] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven wan. In *SIGCOMM*, pages 15–26, 2013.
[11] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: experience with a globally-deployed software defined wan. In *SIGCOMM*, pages 3–14, 2013.
[12] G. Karakostas. Faster approximation schemes for fractional multicommodity flow problems. In *Proc. ACM SODA*, 2002.
[13] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *HotNets*, page 19, 2010.
[14] F. T. Leighton, F. Makedon, S. A. Plotkin, C. Stein, É. Stein, and S. Tragoudas. Fast approximation algorithms for multicommodity flow problems. *Journal of Computer and System Sciences*, 50(2):228–243, 1995.
[15] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter. Traffic engineering with forward fault correction. In *SIGCOMM*, 2014.
[16] J. Liu, A. Panda, A. Singla, B. Godfrey, M. Schapira, and S. Shenker. Ensuring connectivity via data plane mechanisms. In *NSDI*, pages 113–126, 2013.
[17] M. Suchara, D. Xu, R. D. Doverspike, D. Johnson, and J. Rexford. Network architecture for joint failure recovery and traffic engineering. In *SIGMETRICS*, pages 97–108, 2011.
[18] Y. Wang, H. Wang, A. Mahimkar, R. Alimi, Y. Zhang, L. Qiu, and Y. R. Yang. R3: Resilient Routing Reconfiguration. In *SIGCOMM*, 2010.
[19] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.