2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems

Fast Recovery for Single Link Failure with Segment Routing in SDNs

Shishuang Wang, Hongli Xu, Liusheng Huang, Xuwei Yang, Jianchun Liu
School of Computer Science and Technology
University of Science and Technology of China
Hefei, China, 230027
Email:{sshuangw, issacyxw, lyl617}@mail.ustc.edu.cn,{xuhongli, lshuang}@ustc.edu.cn

***Abstract*—Link failure is a major problem that must be addressed, which occurs frequently in large networks. In a software-defined network, the recovery scheme for single link failure can be divided into two categories: proactive and reactive. For reactive scheme, it takes a long time to recover due to the controller participation. Though the proactive scheme can achieve faster recovery, it may consume a large number of forwarding rules, which is typically limited on switches due to expensiveness and power hungry. Therefore, this paper proposes a proactive recovery scheme for single link failure based on segment routing with less forwarding rules. In the proposed scheme, we regard the affected flows through the same link as an aggregated flow and use the link protection methods to achieve link failures recovery. The link protection methods calculate backup paths for each link of the working path, so we propose the backup paths calculation problem and formulate it as an mixed integer non-linear programming model. To solve the problem and avoid link congestion, we design an efficient algorithm to select the most appropriate backup path. In addition, due to the limitation of the maximal number of MPLS labels that the packet header can hold, we design an algorithm to divide the working and backup path into segments to meet the hardware requirements. The simulation results show that the proposed algorithms outperform CAFFE and SPR by 14.7% and 77.1% on recovery time, and outperform CAFFE by 21.5% on forwarding rules consumption. Moreover, the proposed scheme does not cause congestion in post-recovery network.**

***Index Terms*—fast recovery, software-defined networks, single link failure, link congestion avoidance, segment routing**

I. Introduction

In a software-defined network (SDN), network engineers can adjust traffic in a centralized manner. Due to link congestion, bandwidth unavailability, link or node failures, fast data transmission is interrupted frequently. In particular, link failure is a major problem that must be addressed, which single link failure can occur every 30 minutes on average [1]. Considering the flexibility of SDN, SDN-based link failure recovery schemes become very attractive.

Techniques have been devised to mitigate the consequences of link failures by rerouting interrupted traffic from failed link to backup paths. There are two main challenges encountered when performing link failure recovery. The first challenge is how to reduce the recovery time, and the second challenge is how to effectively utilize the limited network resources. In order to solve the two challenges mentioned above, the recovery approach can be divided into two categories: proactive recovery [2] [3] [4] and reactive recovery [5] [6] [7]. In proactive schemes, the forwarding rules of backup paths are pre-installed before any single link fails. After the link fails, switches can steer interrupted flows to backup paths without controller involvement. The most obvious advantage is that they can achieve quickly and timely recovery. The forwarding rules are stored in TCAM, which has limited capacity because of expensiveness and power hungry [8]. In proactive schemes, the forwarding rules of backup paths will consume TCAM resources even if no link failure occurs, which may cause flow table overflow.

In reactive schemes, the backup paths are dynamically calculated after single link fails. Compared with proactive schemes, the reactive schemes consume fewer forwarding rules. However, reactive schemes require more time to recover. In addition to the link failure detection time, the reactive schemes require the controller to recalculate the route of interrupted flows and install the forwarding rules on the corresponding switch. Long-term link failure recovery will cause massive packet loss.

In order to satisfy short recovery time and less forwarding rules consumption at the same time, this paper proposes a proactive fast recovery scheme for single link failure based on segment routing. The main contributions are as follows:

- For the purpose of reducing the forwarding rules consumed, we adopt the segment routing [9] method which steer packets through an ordered list of instructions [10] in packet header and regard the affected flows through the same link as an aggregated flow and establish backup path for the aggregated flow.
- We formulate the backup paths calculation problem and design an algorithm to select the most appropriate backup path to avoid post-recovery network congestion.
- To overcome the limitation of Maximum SID Depth (MSD, the maximal number of MPLS labels that the packet header can hold), we design an algorithm to divide the working and backup path into several segments to meet the hardware requirements.

The paper is organized as follows. In Section II, we introduce background and motivations. In Section III, we introduce network model and formulate backup paths calculation problem in detail. In Section IV, we present the algorithms for

978-1-7281-2058-4/19/$31.00 ©2019 IEEE
DOI 10.1109/HPCC/SmartCity/DSS.2019.00278

backup path selection and path segmentation. In Section V, we show the simulation results. In Section VI, we conclude the paper.

II. Background and Motivation

A. Proactive Schemes

To implement proactive schemes, We can leverage the concept of group table which proposed by Open Networking Foundation (ONF) [11]. Each group entry consists an ordered list of action buckets, and each action bucket contains parameters and actions. The flow entry can forward packets to group entry by group ID. For the purpose of supporting automatic link failures recovery, we leverage the "fast failover" group type, in which each action bucket has a port parameter that controls the entries' liveness, and the group entry will execute the first live action bucket. As shown in Fig. 1, when switch $S1$ receives packets, switch $S1$ will find a matched flow entry, which points to a group entry with group ID 123. The corresponding group entry contains two action buckets: Bucket1 and Bucket2. The two action buckets have their respective associated ports. Under normal circumstances, the packets will be forwarded by the first live Bucket1. If link $S1 - S3$ fails, the port 2 of switch $S1$ changes from "Up" to "Down" and Bucket1 becomes unavailable, so Bucket2 will be automatically applied to the following packets.

In proactive schemes, the backup path is calculated in advance, and the link failure can be recovered quickly. In [2], the authors propose Forward Local Rerouting(FLR) and Backward Local Rerouting (BLR) algorithm to calculate backup paths by leveraging proactive schemes. The algorithms output backup paths with the least number of additional switches, and restrict additional switches to have only one forwarding rule, which can avoid forwarding loops. However, the schemes does not take into account link bandwidth limitations, and may cause link congestion after failure recovery. In [3], the authors propose CAFFE, which supports multiple types of failure recovery, and the controller calculate backup paths by not using heavy-loaded links to avoid potential link congestion. In [4], in order to deal with link failures, the authors propose fast fail-over mechanism, in which controller periodically calculates multiple paths and pro-actively installs flow and group entries. In addition, the authors leverage the fast switch-over mechanism to avoid link congestion. In the above two schemes [3] [4], the authors don't consider the capacity limitation of TCAM, so the proposed schemes may cause flow table overflow, which will cause massive packets loss.

B. Reactive Schemes

In reactive schemes, the backup path is dynamically calculated after link failure occurs. Due to frequent interactions between the switches and the controller, link failure recovery takes longer than proactive schemes. In hybrid SDN networks, the authors of [5] propose an approach that redirect traffic on failed link to SDN switches to guarantee traffic accessibility. In addition, with the coordination between SDN switches, the approach can explore multiple backup paths for fail-over, which can avoid potential link congestion. However, this scheme must depend on underlying IGP protocol, which increases the complexity of data layer. In order to balance recovery time and forwarding rules occupation, the authors of [6] leverage flexible flow aggregation. To avoid link congestion, each interrupted traffic is reassigned to a new reroute path. Therefore, all flows that assigned to the same path are aggregated. In addition, the authors formulate the optimization problem as ILP model and propose CALFR algorithm to solve it. In [7], the authors propose CFR scheme, which contains a 2-stage algorithm. In the first stage, calculating the backup paths of all links. In the second stage, to make better use of network resources, they leverage the network state to adjust the backup paths. In the above three schemes [5] [6] [7], they are essentially reactive solutions, and the recovery time for link failures cannot meet the industrial requirements.

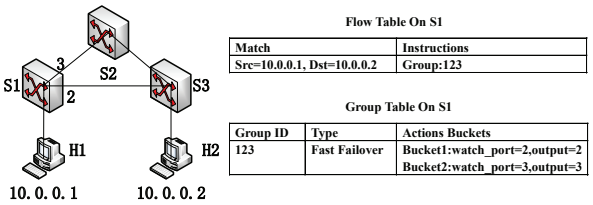

Fig. 1. Leveraging flow and group table to implement link failure recovery. When $S1$ receives packets, it will find a matched flow entry, which points to group entry 123 and packets will be processed by the action buckets. Under normal circumstances, the packets will be forwarded by Bucket1. If link $S1 - S3$ fails, the port 2 of switch $S1$ changes from "Up" to "Down" and Bucket1 becomes unavailable, so Bucket2 will be automatically applied to the following received packets.

TABLE I
Performance Comparison of Different Link Failure Solutions

	TCAM occupation	Recovery Time	Post-recovery Congestion
[2]	Less	Short	Yes
[3]	More	Short	No
[4]	More	Short	No
[5]	Less	Long	No
[6]	Less	Long	No
[7]	More	Long	No

C. Motivation

As shown in TABLE I, we found that the focus of the above solutions are different. In summary, we can compare the performance from the aspects of TCAM occupation, Recovery Time after link failures and Post-Recovery Congestion. Among the three performance metrics mentioned above, we want to design a recovery scheme with less TCAM occupation, short recovery time, and no link congestion.

In segment routing, a route path of flow is configured in the packet header by source node [12], which makes that the controller only needs to install forwarding rules on ingress switch. Through the combination of proactive schemes and segment routing, we can strike a balance trade-off between link

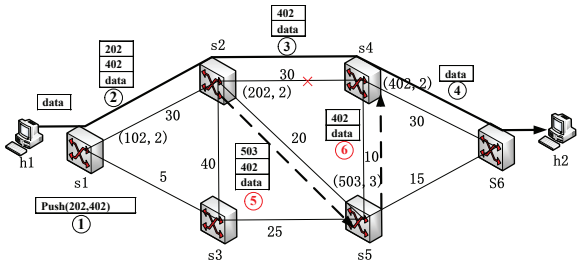

Fig. 2. Example of link failure recovery based on segment routing. The number next to each link is its current traffic load and the link bandwidth is set to 100. We assume that a flow f is from $h1$ to $h2$ and link $s2-s4$ will fail. Before link $s2-s4$ fails, the packets are forwarded through path $\{s1 \rightarrow s2 \rightarrow s4 \rightarrow s6\}$. The proposed scheme selects path $\{s2 \rightarrow s5 \rightarrow s4\}$ as the backup path of link $s2-s4$, in which case the maximal link utilization is minimized to 0.5. Upon link $s2-s4$ fails, switch $s2$ will automatically steer the packets to backup path.

failure recovery time and forwarding rules consumption, and get a scheme with less TCAM occupation and short recovery time. However, due to the limitation of MSD, the number of labels that the switch can push simultaneously is limited, so we need to design an algorithm to segment the routing path. In addition, potential link congestion can be avoided by properly selecting the backup path.

III. Preliminaries and Problem Formulation

A. Network Model

An SDN network topology can be modeled by $G = (V, E)$, where V is a set of switches and E is a set of links connecting switches. Same to previous works [6] [13], we adopt the unsplittable flow mode. In our proposed scheme, the controller calculates backup paths for each link, and the interrupted flows are rerouted through backup path as an aggregated flow to reduce forwarding rules. Since the routing path are encapsulated in packets header, during the recovery process, the upstream node of faulty link will remove the label associated with the failed link and add new labels to packets header based on backup path.

Fig. 2 shows an example of our scheme. We assume that a flow is from $h1$ to $h2$, and link $s2-s4$ will fail during network operation. First, the remote controller calculates the working path P_1 (*i.e.* the bold solid lines in Fig. 2). Based on above assumptions, the controller calculates all six backup paths for link $s2-s4$ in advance. Among the six backup paths that can be selected, the maximal link utilization is minimized to 0.5, so we select the corresponding path (*i.e.* the dotted lines in Fig. 2) as the backup path for link $s2-s4$. The flow and group tables on switches are shown in Fig. 3. In fact, in order to protect link $s2-s4$, only switch $s2$ needs to take the switching action. As long as link $s2-s4$ fails, switch $s2$ automatically steers the packets to backup paths. Since the flow entries of backup paths have been installed in advance, the entire recovery process does not require the participation of the controller. Therefore, the failure recovery time is equal to the failure detection time.

TABLE II
NOTATIONS

Notation	Definition
$G(V,E)$	network topology
V	set of nodes
E	set of links
F	set of flows that need to be rerouted
m	the number of reroute flows
f_j	a flow j, $f_j \in F$
b_j	the bandwidth of f_j
r_j	working path of f_j
l	failed link, $l \in E$
s_l	upstream node of link l
d_l	downstream node of link l
r_l	traffic load on link l
P	backup path set that from s_l to d_l
p_i	the ith backup path, $p_i \in P, 1 \leq i \leq \lvert P \rvert$
n_i	the number of switches in p_i
l_i^z	the zth link of p_i, $1 \leq z \leq (n_i - 1)$
c_i^z	bandwidth capacity of l_i^z
b_i^z	traffic load on link l_i^z before single link failure occurs
a_i^z	traffic load on link l_i^z after recovery
$t_{u,v}^{p_i}$	binary variable. 1 if p_i traverses link (u,v), 0 otherwise
λ	upper bound of link utilization

B. Backup Path Selection

We use $l \in E$ to indicate the failed link. s_l and d_l represent upstream and downstream node of the failed link, respectively. Let F denote the rerouted flows, and use P to denote the set of backup paths that from s_l to d_l. The full notations list is shown in TABLE II. Our objective is to find the maximal available bandwidth path. The formal formulation is presented as follows.

$$\min \lambda$$

subject to the following constraints:

$$t_{s_l,d_l}^{p_i} = 0, \forall p_i \in P \tag{1}$$

$$\sum_{v:(u,v)\in E} t_{u,v}^{p_i} - \sum_{v:(v,u)\in E} t_{v,u}^{p_i} = \begin{cases} 1, & u = s_l \\ -1, & u = d_l \\ 0, & \text{otherwise} \end{cases} \quad \forall p_i \in P \tag{2}$$

$$\sum_{v\in V,(u,v)\in E} t_{u,v}^{p_i} \leq 1, \forall u \in V \tag{3}$$

$$\sum_{j=1}^{m} b_j * t_{s_l,d_l}^{r_j} = r_l, \forall f_j \in F \tag{4}$$

$$b_i^z + r_l = a_i^z, \forall p_i, 1 \leq z \leq (n_i - 1) \tag{5}$$

$$a_i^z / c_i^z \leq \lambda \leq 1, \forall z, p_i \tag{6}$$

Equation (1) denotes that the every backup path must bypass the failed link l. Equation (2) denotes flow conservation,

Flow Table on s1

Match	Instructions
Src=IP1,Dst=IP2	Push_mpls (202, 402),Group:2

Group Table on s1

Group ID	Type	Actions Buckets
2	Fast Failover	Bucket1:watch_port=2,output=2

(a) The Flow and Group Table on Switch s1

Flow Table on s2

Match	Instructions
Mpls=202	Group:2

Group Table on s2

Group ID	Type	Actions Buckets
2	Fast Failover	Bucket1:watch_port=2,pop 202,output=2 Bucket2:watch_port=3,pop 202, push_mpls 503,output=3

(b) The Flow and Group Table on Switch s2

Flow Table on s4

Match	Instructions
Mpls=402	Group:2

Group Table on s4

Group ID	Type	Actions Buckets
2	Fast Failover	Bucket1:watch_port=2,pop 402,output=2

(c) The Flow and Group Table on Switch s4

Flow Table on s5

Match	Instructions
Mpls=503	Group:3

Group Table on s5

Group ID	Type	Actions Buckets
3	Fast Failover	Bucket1:watch_port=3,pop 503,output=3

(d) The Flow and Group Table on Switch s5

Fig. 3. The flow and group table on different switches. Ingress switch $s1$ pushes two MPLS labels and forwards the packets to switch $s2$(①②). Since the outmost label is 202, switch $s2$ processes the packets by group table entry 2. Under normal circumstances, the packets are forwarded by Bucket1, which removing the outmost MPLS label and delivering the packets to port 2(③). Upon link $s2-s4$ fails, the port 2 of switch $s2$ changes from "Up" to "Down", so the Bucket1 become unavailable, the packets will be handled by Bucket2 automatically, which removing the invalid label and add a new one(503). The other related switches forward packets based on MPLS label(④⑥).

which means that the traffic entering the intermediate node u is the same as the traffic leaving u node, and the traffic must completely exit its source node and completely enter its destination node. Equation (3) denotes that the selected backup path should avoid loops. Equation (4) calculates the traffic load through the failed link l. It equals to the sum of bandwidth of flow which passing through the failed link. Equation (5) defines the traffic load on link l_i^z after recovery. Equation (6) denotes the link capacity utilization constraint.

For the purpose of solving the optimization problem in polynomial time, we design an efficient algorithm (as show in Algorithm 1)).

C. Working and Backup Path Segmentation

As mentioned in Section II, in segment routing, the path which the packets must pass through is encoded as label stack in packet header by ingress switch, and the packet forwarding is implemented by the label stack. However, due to the MSD constraint, we can not push all link labels to the packet header if routing path is too long, so we should divide the working and backup paths into several segments to meet the hardware requirements. The detailed solution is shown in Algorithm 2.

IV. Algorithm Design

A. Determining Selected Backup Path Set

In the proposed algorithm, in order to avoid link congestion, the remote controller adopts a greedy-based strategy, which selects the path with the lowest link utilization from all backup paths of the protected link as the *Selected Backup Path*.

The detailed design is shown in Algorithm 1. The overall process can be divided into four steps: (1) Calculating paths between two nodes of each link (lines $1-7$). Function *AllSimplePath* (line 6) is based on depth-first search, which establishes a stack with source node and iteratively search all paths between the two nodes of each link. (2) Calculating the total traffic through each link (lines $8-14$). For the working path $W_f \in W$, it is checked whether it traverses link e or not, and if it passes, the bandwidth of flow f will be added to R_e. (3) Updating the backup path set (lines $15-19$). For all backup paths of link e, if the remaining available bandwidth is smaller than the total traffic on link e, the corresponding path will be removed. (4) Selecting the most appropriate path (lines $20-29$). The variable θ (line 22) denotes the upper bond of link utilization, which is a constant, generally $0 \leq \theta \leq 1$, and we set $\theta = 0.8$ in this paper. The selection process is mainly based on greedy strategy, which selects the path with the least link utilization.

B. Segmenting Backup and Working Path

In our proposed schemes, all the links that the packets will go through are encapsulated in packet header. Accordingly, the label stack will be too large to comply with MSD constraint. Therefore, we design an algorithm to segment the working and backup paths.

The details are shown in Algorithm 2. In lines $1-2$, the algorithm initializes the variables. In lines $3-6$, the algorithm marks the switches. For some switches in the path, the algorithm marks it as *Contact Switch*. In lines $7-13$, the algorithm constructs the sub-paths based on the marked switches. Starting from the ingress switch, a sub-path will be formed between every two *Contact Switch* (Lines $11-13$).

C. Algorithm Complexity Analysis

Let m denote the number of flows that need to be rerouted. In Algorithm 1, the complexity of function *AllSimplePath* is $O(|E|+|V|)$, where $|E|$ is the number of edges and $|V|$ is the

Algorithm 1 Algorithm for Backup Path Selection

Input: network topology $G(V, E)$, working path set W
Output: the selected backup path of each link $e \in E$

Step 1: Calculating paths between two nodes of each link

$B \Leftarrow \{\}$; // the set of reachable paths of all links
for all $e \in E$ **do**
  $B_e \Leftarrow \{\}$; // the set of reachable paths of link e
  $Src_e \Leftarrow$ upstream node of link e;
  $Dst_e \Leftarrow$ downstream node of link e;
  $B_e \Leftarrow AllSimplePath(Src_e, Dst_e, G, cutoff = D)$;
  $B \Leftarrow B \cup \{B_e\}$;

Step 2: Calculating the total traffic through each link

$R \Leftarrow \{\}$; // the set of traffic passing through each link
for all $e \in E$ **do**
  $R_e \Leftarrow 0$; // the total traffic passing through link e
  for all $W_f \in W$ **do**
    if W_f traverse e **then**
      $R_e \Leftarrow R_e + b_f$;
  $R \Leftarrow R \cup \{R_e\}$;

Step 3: Updating the backup path set

for all $B_e \in B$ **do**
  for all $b \in B_e$ **do**
    $b_{avi} \Leftarrow AvailableBandwidth(b)$;
    if $b_{avi} < R_e$ **then**
      remove b from B_e;

Step 4: Selecting the most appropriate path

$B^{selected} \Leftarrow \{\}$; // the backup path selected by all links
for all $B_e \in B$ **do**
  $LU_{min} \Leftarrow \theta$; // temporary path minimum link utilization
  $B_e^{selected} \Leftarrow \{\}$; // the backup path selected by link e
  for all $b \in B_e$ **do**
    $b_{link\_uti} \Leftarrow LinkUtilization(b)$;
    if $b_{link\_uti} < LU_{min}$ **then**
      $B_e^{selected} \Leftarrow b$;
      $LU_{min} \Leftarrow b_{link\_uti}$;
  $B^{selected} \Leftarrow B^{selected} \cup \{B_e^{selected}\}$;
return $B^{selected}$

number of nodes, so the complexity of first step is $O(|E|^2 + |E||V|)$. The complexity of other steps in algorithm 1 are $O(m|E|)$, $O(|E|^2)$, and $O(|E|^2)$, respectively. Therefore, the overall complexity of Algorithm 1 is $O(|E|^2+|E||V|+m|E|)$. The complexity of Algorithm 2 is $O(|V|)$.

V. Simulation Results

A. Simulation Setting

To evaluate the proposed scheme, we use Ryu [14] as the central controller and Mininet [15] to emulate the tested topologies. All simulation experiments are conducted on Ubuntu 16.04 in VMWare workstation.

We use three different topologies to evaluate the proposed segment-routing based scheme(SRBS): an example topology in Fig. 2, an actual topology USNET [16] and a Fat-tree

Algorithm 2 Algorithm for Routing Path Segmentation

Input: working or backup routing path p, first segment length l_{fs}, other segment length l_{os}
Output: the segment(sub-path) list of the path p

Step 1: Initialization

$\overline{SL} \Leftarrow \{\}, \overline{TNL} \Leftarrow [\,]$; //segment and temporary node list
$N \Leftarrow$ the number of switches belong to path p;

Step 2: Marking the Contact Switch in the path

$index \Leftarrow 1$;
while $index \leq N$ **do**
  Mark the $SWITCH_{index}$ as the *Contact Switch*;
  Update $index$ based on l_{fs} and l_{os};

Step 3: constructing sub-paths

for $i = 1; i \leq N; i++$ **do**
  Add the $SWITCH_i$ to the end of $\overline{TNL}$;
  if $i == N$ **then**
    Add $\overline{TNL}$ to the end of $\overline{SL}$ and clear $\overline{TNL}$;
  else if $\overline{TNL}_{first}$ is *Contact Switch* **and** $\overline{TNL}_{first} \neq \overline{TNL}_{end}$ **and** $\overline{TNL}_{end}$ is *Contact Switch* **then**
    Add $\overline{TNL}$ to the end of $\overline{SL}$ and clear $\overline{TNL}$;
    $i \Leftarrow i - 1$; // move i one step backward
return $\overline{SL}$

topology (k=4) for data centers. We let the link capacity equals to 50 Mbps, and the traffic bandwidth is uniformly distributed in the range of [3 Mbps, 5 Mbps]. We compare SRBS with Shortest Path Restoration(SPR) [17] and CAFFE [3].

B. Forwarding Rules

We now evaluate the number of forwarding rules that are required after single link failure. The result in Fig. 4, 5, 6 are average values of five independent random experiments. The host is randomly connected to the switch. The vertical axis denotes the average forwarding rules on each switch.

As shown in Fig. 4, we can find out that SPR outperforms SRBS by 33.3% on forwarding rules consumption, and outperforms SRBS by 35.1%. In SPR, the backup path is only recalculated after single link failure, so it consumes fewer forwarding rules than the proactive one. Compared with CAFFE, SRBS outperforms by 21.5% on forwarding rules consumption. This is because that SRBS uses segment routing to encapsulate path information in the packet header. Each time an encapsulation is performed, the packet can be forwarded directly in the next few hops with consuming only one forwarding rules. We did the same experiment on USNET topology and got similar results (As shown in Fig. 5).

However, in the Fat-tree topology, the results are somewhat different. When there are fewer hosts, the forwarding rules required by SRBS is more than that of CAFFE. As the number of hosts increases, the forwarding rules consuming in SRBS is approximately the same as that in CAFFE(As shown in Fig. 6).

C. Recovery Time

Fig. 7 shows the average recovery time. The result are the average values of five independent experiments. To get

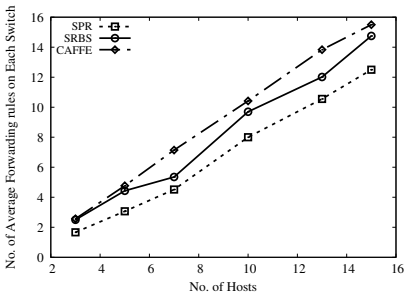

Fig. 4. Forwarding rules in Example topo

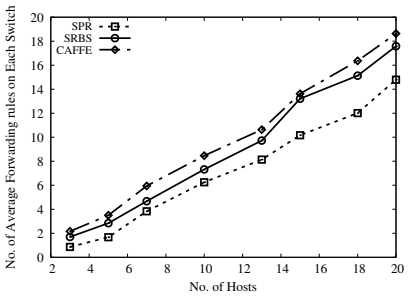

Fig. 5. Forwarding rules in USNET topo

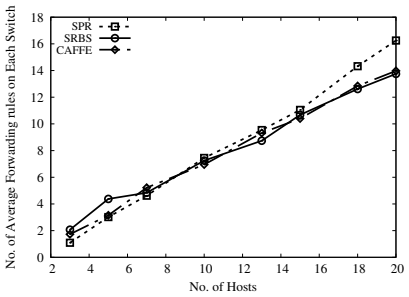

Fig. 6. Forwarding rules in Fat-tree topo

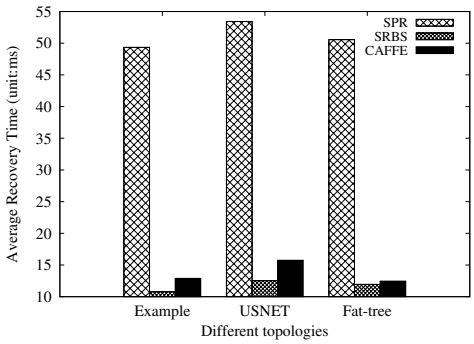

Fig. 7. Average recovery time of different schemes(2 traffic flows)

the recovery time, we use the ping to generate the packet, and randomly select a link of routing path to disconnect it, and use Wireshark [18] to monitor the packet at destination host. The recovery time is obtained by taking the first packet reception time after the link failure occurs minus the last packet reception time before the link failure occurs.

As shown in Fig. 7, SRBS consumes the least failure recovery time, and it outperforms SPR by 77.1%, and outperforms CAFFE by 14.7% in average. This is because SPR is a reactive scheme that requires controller involvement for fail-over, which makes reactive scheme unacceptable [19]. Compared with CAFFE, SRBS slightly reduces recovery time. The reason is that they are both proactive schemes, and SRBS consumes less forwarding rules.

VI. Conclusion

In this paper, we leverage segment routing to implement fast recovery for single link failure in SDNs. We use the "fast fail-over" group table type to achieve proactive failure recovery and consider congestion avoidance when calculating the backup path. In addition, due to the limitation of MSD, and each bucket action of the group table entry can only push one MPLS label once, we design an algorithm to divide the working and backup path into segments to meet the hardware requirements. Extensive emulation results show that SRBS can achieve faster recovery and consume less forwarding rules than previous link failure recovery schemes.

References

[1] D. Turner, K. Levchenko, A. C. Snoeren, and S. Savage, "California fault lines: understanding the causes and impact of network failures," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 315–326, 2011.

[2] P. M. Mohan, T. Truong-Huu, and M. Gurusamy, "Tcam-aware local rerouting for fast and efficient failure recovery in software defined networks," in *Global Communications Conference (GLOBECOM), 2015 IEEE*. IEEE, 2015, pp. 1–6.

[3] Z. Zhu, Q. Li, S. Xia, and M. Xu, "Caffe: Congestion-aware fast failure recovery in software defined networks," in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2018, pp. 1–9.

[4] Y.-D. Lin, H.-Y. Teng, C.-R. Hsu, C.-C. Liao, and Y.-C. Lai, "Fast failover and switchover for link failures and congestion in software defined networks," in *Communications (ICC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–6.

[5] C.-Y. Chu, K. Xi, M. Luo, and H. J. Chao, "Congestion-aware single link failure recovery in hybrid sdn networks," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE, 2015, pp. 1086–1094.

[6] Z. Cheng, X. Zhang, Y. Li, S. Yu, R. Lin, and L. He, "Congestion-aware local reroute for fast failure recovery in software-defined networks," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 9, no. 11, pp. 934–944, 2017.

[7] L. Wang, L. Yao, Z. Xu, G. Wu, and M. S. Obaidat, "Cfr: A cooperative link failure recovery scheme in software-defined networks," *International Journal of Communication Systems*, vol. 31, no. 10, p. e3560, 2018.

[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 *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 254–265.

[9] J. Tantsura, I. Milojevic, E. Crabbe, S. Previdi, M. Horneffer, A. Bashandy, R. Shakir, C. Filsfils, B. Decraene, S. Ytti *et al.*, "Segment routing architecture," 2013.

[10] "Segment routing," http://www.segment-routing.net/.

[11] "Openflow networking foundation," https://www.opennetworking.org/.

[12] F. Lazzeri, G. Bruno, J. Nijhof, A. Giorgetti, and P. Castoldi, "Efficient label encoding in segment-routing enabled optical networks," in *Optical Network Design and Modeling (ONDM), 2015 International Conference on*. IEEE, 2015, pp. 34–38.

[13] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates," in *Proceedings of the 2014 ACM conference on SIGCOMM*. ACM, 2014, pp. 539–550.

[14] "The ryu sdn framework," http://osrg.github.io/ryu/.

[15] "The mininet platform," http://mininet.org/.

[16] "The usnet service location," http://www.usnet-1.com/locations.php.

[17] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Enabling fast failure recovery in openflow networks," in *2011 8th International Workshop on the Design of Reliable Communication Networks (DRCN 2011)*. IEEE, 2011, pp. 164–171.

[18] "Wireshark," https://www.wireshark.org/.

[19] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Openflow: Meeting carrier-grade recovery requirements," *Computer Communications*, vol. 36, no. 6, pp. 656–665, 2013.