HAL archives-ouvertes.fr

Segmented Source Routing for Handling Link Failures in Software Defined Network

Sharvari Komajwar, Turgay Korkmaz

▶ **To cite this version:**

Sharvari Komajwar, Turgay Korkmaz. Segmented Source Routing for Handling Link Failures in Software Defined Network. International Conference on Wired/Wireless Internet Communication (WWIC), Jun 2018, Boston, MA, United States. pp.146-158, 10.1007/978-3-030-02931-9_12 . hal-02269737

HAL Id: hal-02269737

https://hal.inria.fr/hal-02269737

Submitted on 23 Aug 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution| 4.0 International License

Segmented Source Routing for Handling Link Failures in Software Defined Network

Sharvari Komajwar Turgay Korkmaz

Department of Computer Science
The University of Texas at San Antonio
San Antonio, Texas-78249
{sharvari.komajwar,turgay.korkmaz}@utsa.edu

Abstract. When a link fails in Software Defined Networks (SDN), the flows that use the failed link need to be rerouted over other paths. To achieve this rerouting task, researchers have proposed reactive and proactive recovery approaches. In reactive approach, upon failure, SDN controller computes new paths for the affected flows and installs them on demand. In proactive approach, the SDN controller pre-calculates backup paths and installs them on the switches in advance. While proactive approach minimizes packet loss and delay, it introduces a new problem, namely excessive usage of limited TCAM memory at SDN switches. In this paper, we consider two promising techniques (namely source routing and segment routing), and propose a new proactive technique called Segmented Source Routing (SSR). SSR uses source routing but in a segmented manner: one from the failure detecting node to an emergency node and one from emergency node to the destination. After addressing various challenges in placing emergency nodes and assigning emergency nodes to flows, our simulations shows that SSR maintains the same level of performance of pure source routing while significantly reducing the memory overhead, computation overhead, and the packet sizes as it shortens the source routes and avoids storing them at every node.

Keywords: SDN · Link Failure · Source Routing · Segment Routing.

1 Introduction

One of the key issues in SDN is how to re-route the flows on a failed link through other paths. Formally, this problem can be stated as follows.
Definition 1 *Link Failure Handling (LFH) Problem*: Consider a network that is represented by a directed graph $G = (V, E)$, where V is the set of nodes/switches and E is the set of links. Let $n = |V|$ be the number of nodes in the network and $m = |E|$ be the number of edges in the network. Each link $(u, v) \in E$ is associated with a cost parameter $c(u, v)$. Suppose the SDN controller accurately maintains this network state information and uses it to compute the shortest paths for each flow request that goes from a source node s to a destination node d. Suppose there are F flows passing through a link (u, v). If that link fails, all

the flows using the failed link (u, v) will be affected as the packets belonging to these flows get lost and/or delayed. Given a failed link (u, v), the LFH problem is how to quickly and efficiently reroute all the affected F flows to other paths so that we can avoid or minimize the packet loss and delay.

In response to addressing the LFH problem, researchers have proposed various recovery mechanisms in the literature [12, 8, 6, 10, 7, 1, 9, 11]. The existing techniques are mainly of two types: Reactive (or path restoration) [12] and Proactive (or path protection) [8, 7, 1, 9, 11, 10]. In the case of *reactive* solutions, the controller needs to be informed about the link failure. Upon receiving link failure notification, the controller determines a new path and updates all the switches related to the new path. The main advantage of the reactive approach is the fact that the underlying switches do not need to store any extra backup paths or state information besides the primary paths. Moreover, the found path will be the best one under the given network state information. However, due to the extra times required for conveying the link failure information to the controller, computing new paths, and consistently updating/installing new forwarding rules [4, 5], the reactive techniques loose and/or delay many packets until the recovery is completed. To minimize the recovery time, Sharma et al. in [12] have proposed an improvement where the controller pre-calculates all the backup paths and use them on demand to reroute the affected flows. However, due the dominance of delays in conveying link failure and updating new rules consistently, reactive techniques would be still very slow to avoid packet loss and/or delay in practice. As a mater of fact, Sharma et al. demonstrates that it is hard to obtain less than 50ms recovery times when using a controller-based restoration approach [13].

To avoid (or minimize) packet loss and delay during recovery time, the researchers have considered *proactive* techniques [8, 7, 1, 9, 11, 10]. The basic idea here is to have a backup path readily available so that the packets from the affected flows can quickly be re-routed without waiting for the controller's intervention. The performance and the cost of this approach depends on how to determine and maintain backup paths. At one extreme, while installing the primary path for a flow, the controller computes and installs a backup path per flow from every node on the primary path to the destination. Clearly, this extreme version of proactive approach significantly speeds up the response time and thus totally avoids the packet loss and delay as the backup path is readily available at each node. However, this improvement comes at the cost of excessive memory usage for storing and maintaining additional backup paths per flow on the underlying switches, where the TCAM memory is limited and consumes significant amount of energy [10, 7, 9, 8]. For example, if we have F flows going through a link (u, v), then we have to maintain at least $2F$ flow entries at node u, which will be a significant memory overhead as F increases. Therefore, it is deemed necessary to limit the number of backup paths for efficient use of memory space while being able to quickly reroute the affected flows.

With this in mind, researchers have investigated different proactive techniques. For example, Capone et al. have considered utilizing the cranckback routing idea to avoid maintaining backup paths at each node [1]. In this case,

some backup paths are computed from some selected nodes and installed through the network. Upon a link failure, the failure detecting node uses cranckback routing to send the packets back. When the packets reach a node with a backup path, that node re-routes the traffic. While this reduces the number of backup paths, the reverse paths might be longer or congested, causing delays and loss. Sgambelluri et al. have proposed to use backup paths per destination rather than per flow [11]. While these approaches reduce the number of backup paths and saves some memory, they may still use significant amount of TCAM space for storing and maintaining the backup paths or the backup path might not be the best one.

To minimize the excessive TCAM memory overhead while using better paths, researchers have considered new techniques based on *source routing* [8, 10] and *segment routing* [6]. In source routing, the controller calculates the shortest paths from each source node to all other destination nodes and stores the complete paths at each source node. When a particular flow is created, its source node checks the destination and inserts the whole path to that destination into the packet header. In SDN, VLAN tags can be pushed into the header to store the whole path information, which contains either the IDs of nodes along the path or just port IDs on each node of the path [8, 14].

In [8], Huang et al. have proposed to use source routing for maintaining backup paths. In source routing, every switch u on the primary path of a flow stores the complete backup paths to its destination rather than installing it throughout the network. To do this, the controller eliminates each link (u, v) on the primary path at a time, and computes the shortest paths from u to the destination of the flow. It then stores the corresponding path on switch u. So, if F flows are passing through u, then that switch has to store F source routes (note that these backup paths are not installed throughout the network). When there is a failure on link (u, v), node u will quickly detect the link failure and (without waiting for the controller) it will reroute the incoming packets by inserting the pre-stored source route from u to the destination into the packet header and forward it to the next node. Clearly, source routing minimizes the number of flow entries in the switches. But the length of each entry increases since the whole backup path is stored per flow [8]. Effectively, each switch needs to maintain F many backup paths, which will be costly as F increases. Moreover, the authors in [8] propose to update these backup paths after some interval of time based on the current status of the network, which further increases the computation overhead on the controller. Another issue with pure source routing is that it increases the packet size as the whole backup path is included into the packets.

Segment routing (SR) [3] is similar to loose source routing (an IP option to record the set of routers that a packet must visit). In contrast to the pure source routing, SR inserts the IDs of a few nodes into the packet header and then tries to send a packet to the next node by using the paths determined and maintained by the underlying routing protocol. The path between the consecutive nodes in the header is called a segment. In [6], authors have considered using 2-segment

routing to deal with LFH problem. This approach limits the number of node IDs in the header and the number of entries in the SDN flow tables. However, it relies on the underlying protocol to compute new paths for the segments to deal with the link failure. Unfortunately, when the failed link is on the segment that needs to be used, this approach can still cause packet loss and delay until the underlying routing protocol finds new paths, as in the reactive approach. Moreover, in segment routing when link fails, all the paths on intermediate nodes need to be updated consistently to guarantee loop-free and black-hole-free routing.

In this paper, we propose a new link failure handling technique called Segmented Source Routing (SSR) by merging the best of the source routing and segment routing. As in the source routing, SSR includes the path information into the packet headers so that we can avoid memory overhead by not installing backup paths through the network. However, to further minimize the number of source routes maintained at each node and to minimize the length of the paths included in the packet headers, SSR does this in a segmented manner and per destination rather than per flow. In SSR, we first identify some nodes as emergency nodes. The controller then pre-computes the shortest paths from every node to the emergency nodes and from emergency nodes to every node. We store these paths at their respective source nodes. When a flow request arrives, the controller determines a primary path and installs it as usual. In contrast to the pure source routing, which computes a backup path from each node on that primary path [8], our SSR approach simply determines which emergency node (say node e) to use at each node u. For each flow, this information is maintained in the flow table at node u. Upon detecting a link failure, node u simply inserts the source route from u to e into the packet headers of the affected flows and sends them towards the corresponding emergency node e. Upon receiving such a packet, emergency node e inserts the source route from e to d and sends it towards d. In contrast to the pure segment routing, we do not rely on the underlying routing protocols to find paths for segments. Instead, we determine each segment using source routing. In the following sections, we will further describe the proposed SSR framework and address the challenges in it.

The rest of the paper is organized as follows. Section 2 gives the overview of the proposed SSR framework. Section 3 addresses the challenges in SSR. Section 4 presents the performance evaluation using simulation. Section 5 talks about future work and concludes the paper.

2 The Proposed Segmented Source Routing Framework

To quickly and efficiently respond to the link failures, we propose a new technique called Segmented Source Routing (SSR). In essence, SSR is similar to source routing. However, instead of including the whole source route into packet headers, SSR divides the path into two segments and include the source route for each segment into packet headers one at a time. To be able to that, SSR designates some nodes in the network as the *emergency* nodes. Then, SSR makes every emergency node store the list of source routes to all destinations while making

every other switch/node store only the source routes to all the emergency nodes. Since the number of emergency nodes is expected to be significantly less than n, we will avoid significant memory overhead when storing source routes. Moreover, instead of storing source routes per flow as in [8], SSR stores source routes per destination. This way SSR minimizes both the length of the path information included into packet headers and the number of source routes stored. This will significantly reduce memory overhead, particularly when the number of flows F increases.

When installing the primary path for a particular flow, the SDN controller needs to determine which emergency nodes to contact from each node on the primary path. Emergency node information is included as part of the flow entry at each node on the primary path. So when link (u, v) fails, node u can detect the failure and identify the emergency node for each flow passing through this link. Accordingly, node u inserts the first segmented source route from u to the corresponding emergency node e into the packet header and sends it. Upon receiving such a packet, the emergency node e inserts the second segmented source route from the emergency node to the destination.

For example, Fig. 1 shows a link failure scenario where primary shortest path from the source $S4$ to destination $D2$ is $\{S4, S9, S13, S18, D2\}$. Suppose node

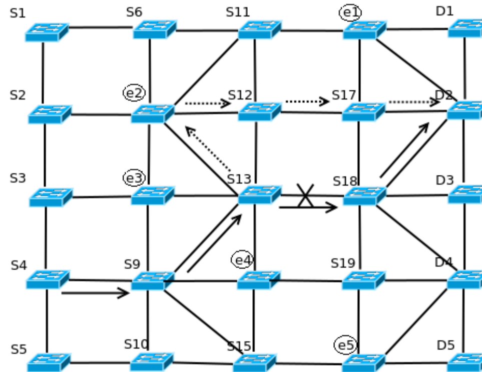


Fig. 1. An Example for Link Failure Handling.

$e2$ is the emergency node for that flow at node $S13$ and the link $(S13, S18)$ fails. In this case, the switch $S13$ detects the link failure and inserts the first segmented source route $\{S13, e2\}$ into the packets of that flow and sends them to the emergency node $e2$. Upon receiving these packets, the emergency node $e2$ inserts the second segmented source route $\{e2, S12, S17, D2\}$ into the packets and sends them to the destination node $D2$.

Instead of using pure source routing per flow as in [8] or the pure segment routing as in [3], we combine the best of these two mechanisms as segmented source routing, which pre-computes and installs two segments of source routes:

$segment(u, e)$ from every node u to each emergency node e; $segment(e, d)$ from every emergency node e to every destination node d. Then, when installing a primary path, the SDN controller decides which emergency node to use for each node on the primary path, and includes the ID of the selected emergence node in the flow entry. So, when a link fails, the failure detecting node u simply includes the corresponding $segment(u, e)$ into the packets of the affected flows. Receiving emergency nodes include the corresponding $segment(e, d)$ into the packets and send them.

The key goals of SSR are to (a) reduce the packet size by including shorter source routes into the packets, (b) minimize the number of source routes maintained at each node, and (c) avoid the unnecessary path computations during the installation of the primary path. While achieving these goals, the proposed SSR method should maintain the same level of performance as the pure source routing. Using simulations, we show that SSR has almost the same performance in terms of the cost of the backup paths used and the link utilization, while significantly reducing the packet size, minimizing the number of source routes at each node, and avoiding the unnecessary path computations. We should also note that segmented-source routes are computed in a loop-free and black-hole-free manner and can be changed independently without worrying about a network wide convergence of the underlying routing protocol that maintains the paths for the segments in the pure segment routing.

3 Challenges and Solutions in SSR

In this section, we discuss three crucial challenges that need to be addressed for the SSR technique to efficiently handle link failures. Specifically, we consider: (a) how to select and place emergency nodes, (b) how to compute the segmented source routes from each node to emergency nodes and from each emergency node to all destinations, and (c) which emergency node to assign to which flow at each node on the primary path.

3.1 Emergency Node Selection and Placement Problem

Selection of emergency nodes and their placement play an important role in improving the performance of proposed SSR technique. Intuitively, emergency nodes need to be distributed evenly in the network so that a given node can access one of the emergency nodes with minimum number of hops. At the same time, the emergency node should be able to access the destination with minimum number of hops. In this paper, we will content with randomly selecting the desired number of nodes as emergency nodes to achieve even distribution. In the future, we will investigate how to optimally place them to further improve the performance.

3.2 Segmented Source Route Calculation

Once the controller selects emergency nodes in the network, it calculates the shortest paths for two segments. For the first segment, it computes the shortest paths from every node to emergency nodes and stores these as source routes at each node. For the second segment, it computes the shortest paths from every emergency node to all destinations and stores these as source routes at each emergency node.

We compute the shortest paths based on the same cost parameter used for the pure source routing in [8]. Specifically, the cost of a link (u, v) is determined as follows:

$$c(u, v) = \frac{1}{1 - \rho(u, v)} \tag{1}$$

where $\rho(u, v)$ represents the utilization of link (u, v) and computed using

$$\rho(u, v) = \frac{D(u, v)}{B(u, v)} \tag{2}$$

where $B(u, v)$ is the bandwidth capacity of link (u, v) and $D(u, v)$ is the total demand of the flows using link (u, v). In the future, we plan to also consider different cost parameters that take into account the interference of backup paths on the primary paths.

3.3 Per Flow Emergency Node Assignment

In the case of a link failure, the failure detecting node needs to know which emergency node to contact for each affected flow passing through the failed link. This decision should be made by the SDN controller, which determines a primary path and installs the necessary flow table entries as usual. In the pure source routing, the controller computes the shortest path from each node u on the primary path to destination d as a backup path per flow, and stores that path as the source route at node u. In contrast, SSR assigns an emergency node for each node u on the primary path and saves this information as part of the flow entry at node u. Since our segmented-source routes are pre-computed and stored at each node per destination, the controller is not overloaded with backup path computations per flow or sending these paths to each node while installing the primary path. This way SSR significantly reduces the memory requirements at each node while also decreasing the computation and communication overheads at the controller. Such reductions in the computational load and chattiness of the controller will significantly improve its responsiveness and performance.

The selection of emergency node at each node is an important decision as it will impact the overall performance when there is a link failure. So the controller needs to carefully select the emergency node at each node on the primary path. We formally define this problem as follows:

Definition 2 *Emergency Node Assignment (ENA) Problem*: Consider the network model given in Definition 1. Let R be the set of emergency nodes, where

$R \subseteq V$. Suppose the controller has already computed and stored the shortest paths (source routes) from each node $u \in V$ to the emergency node $e \in R$; and from each emergency node $e \in V$ to every node $u \in V$. Upon receiving a request for a flow going from s to d, the controller finds the shortest path (the primary path) p_{sd} as usual. Given R, p_{sd} and the pre-computed segmented-source routes, the EAN problem is to find/assign the best emergency node $e \in R$ for each node $u \in p_{sd} = \{s, \ldots, u, v, \ldots, d\}$ such that the link (u, v) is not included in source routes denoted by $segment(u, e)$ or $segment(e, d)$.

Ideally, we would like to select an emergency node $e \in R$ for each node $u \in p_{sd}$ such that the sum of the cost of $segment(u, e)$ and the cost of $segment(e, d)$ is minimum. However, since we would like to also minimize the packet size by including a source route with minimum number of hop IDs, we should select the segments containing less number of hops. To minimize both hop count and the cost, we propose to select the emergency node $e \in R$ for each node $u \in p_{sd}$ based on the following objective function:

$$\min_{\forall e \in R} \{C(u, e) * H(u, e) + C(e, d) * H(e, d)\} \tag{3}$$

where $C(u, e)$ and $C(e, d)$ represent the total costs and $H(u, e)$ and $H(e, d)$ represents the hop counts of $segment(u, e)$ and $segment(e, d)$, respectively.

Regarding computational complexity, for each node u, this optimization can simply be done in $O(|R|)$ while the pure source routing requires the execution of the shortest path algorithm with $O(|E| + |V| log|V|)$. Note that $|R| << |V|$.

While the above heuristic finds an emergency node quickly in most cases, it is possible that the source path from u to e or the source path from e to d might include the failed link. We call such a path as a non-safe path. So, the controller needs to eliminate such non-safe paths by simply checking it as follows. Suppose we have a link (u, v) on the computed primary path. So when selecting the emergency node e for node u, the controller needs to make sure that the link (u, v) is not part of the source route from u to e or from e to d. Note that since the controller has all the source routes that are stored in the switches, it can do this locally. There is no communication between the controller and switches. In our simulations, we always find a safe path. But in a rare case, if there is no safe path through any emergency node, then we can use the pure source routing for that case only.

3.4 Computation overhead and Memory consumption

The major goals of the proposed SSR method is to reduce the computation overhead on the controller and memory consumption on the switches. In [8], while installing the flow entries for a flow, the controller calculates a backup path (source route) for the possible failure of each link (u, v) on the primary path and stores that source route from u to the destination d on the switch u. Moreover, to keep these backup paths up-to-date, the controller runs a modified Dijkstra for each switch after a particular interval, which further increases the computation overhead on the controller and the chattiness between the controller and

the switches. Since the backup paths are maintained per flow, each flow entry at node u has to contain the full source route from u to d, causing significant memory overhead. In contrast, the proposed SSR method pre-computes the shortest path per destination as in the segment routing. In contrast to segment routing, SSR stores the shortest paths as source routes. So these source routes can be independently computed or updated without relying on the underlying routing protocols to determine and install them in a consistent manner as the pure segment routing does.

Compared to the pure source routing, the proposed SSR approach involves much less computation overhead and memory consumption. Another key advantage of SSR is that since it includes a segment rather than the whole source route into the packets, it decreases the packet size, resulting in efficient use of resources and faster transfer. Despite these advantages, one natural question is to find out how SSR performs in terms of other measures. In the next section, we compare SSR against the pure source routing in terms of the cost of the backup paths used and the level of increase in link utilization after the link failure. Clearly, the pure source routing would be better as it uses per flow backup paths. However, our simulations show that the proposed SSR closely achieves the same performance while using less resources and computation time.

4 Performance Evaluation

4.1 Simulation Setup

We compared our proposed SSR method against the pure source routing (PSR) in [8]. We implemented both methods in Python and compared them by using the realistic network topology shown in Figure 2, which is modified from ANSNET [2].

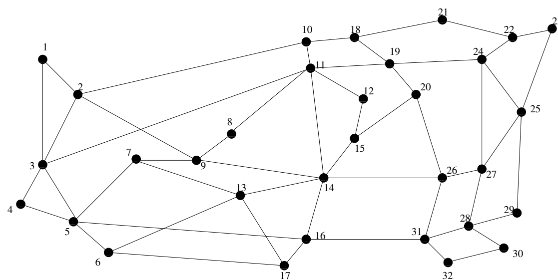


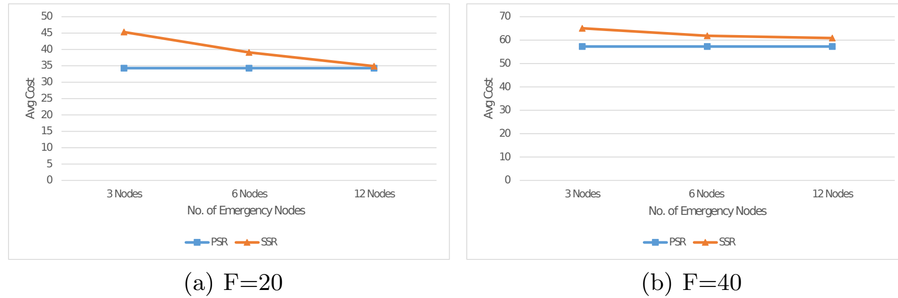
Fig. 2. Topology with n=32, m=110

Our topology (shown in Figure 2) has n=32 nodes and m=110 directed edges. We first randomly select the bandwidth for each link from uniform(10, 20). We randomly select the source node s from uniform(1, 6), the nodes on the left,

while selecting the destination d from uniform(23, 32), the nodes in the right. To test different load, we used two set of experiments with 20 flows and 40 flows. Respectively, the demand of each flow is randomly selected from the range uniform(1,6) and uniform(1,4). For the proposed SSR, we varied the number of emergency nodes as 3, 6, 12 and selected the given number of emergency nodes randomly. After installing the given set of flows, we do not generate any new flow. But we fail randomly selected links one at a time and re-route the affected flows through backup paths.

As the performance measures, we consider (a) the cost of the backup paths at the time of link failures, (b) the increase in link utilization because of using backup paths, and (c) the number of hop IDs included into the packets (hop count). We repeated each experiments 10 times and took their averages.

Average Cost Fig. 3 shows the average cost of the backup paths used by SSR and PSR with different number of flows while varying the number of emergency nodes. The cost of a backup path is calculated using (1) at the time of using that backup path. As shown in Fig. 3, the performance of SSR gets very close to that of PSR as the number of emergency nodes increase from 3 (≈10% of all nodes) to 6 (≈20% all nodes). With 12 emergency nodes (≈40% all nodes), the average cost of backup paths provided by SSR is almost the same as that of PSR.

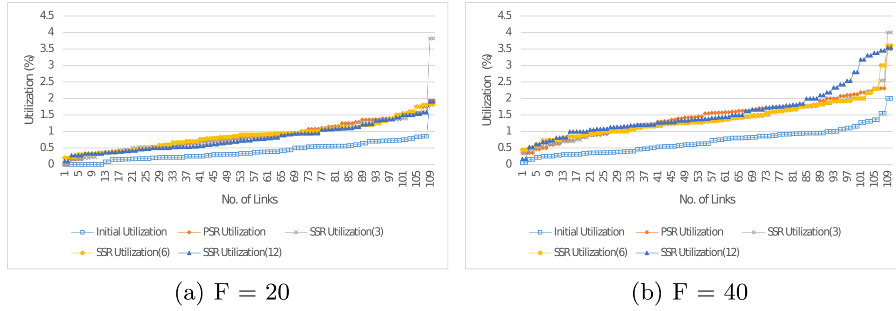


(a) F=20 (b) F=40

Fig. 3. Average cost of backup paths at the time of link failures.

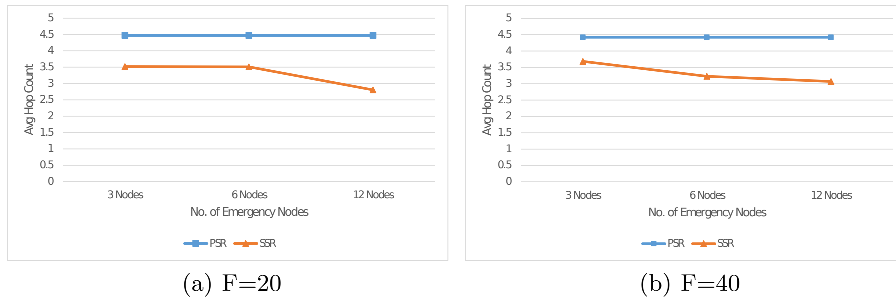
Utilization Fig 4 shows the increase in link utilization for PSR and SSR with 3, 6 and 12 emergency nodes after a failure happens. In the case of 3 emergency nodes, since all the traffic of backup paths goes through these 3 emergency nodes, SSR with 3 emergency nodes makes a few links heavily loaded. But as the number of emergency nodes increases, SSR is able to distribute the load as the PSR does and provide similar link utilization distributions.

Average Hop Count One of the main advantages of the SSR method is the reduction of hop count (i.e., the number of hops included into the packets). As



(a) F = 20 (b) F = 40

Fig. 4. Link utilization distribution before and after link failures.



(a) F=20 (b) F=40

Fig. 5. Average Number of hops carried by the packet

shown in the Fig. 5, SSR method reduces the number of hop information inserted into the packets, particularly as the number of emergency nodes increases.

From Fig. 5, it can be seen that when the number of emergency nodes are $\approx$40% of the total number of nodes, each packet carries approximately 50% of hop information carried by the packets in PSR. As shown in Fig. 5(b), even when the network is highly loaded and there are 6 emergency nodes average hop count is almost 50% of PSR. The main advantage of this reduced hop count is to reduce the packet size, which improves transmission time and avoids unnecessary usage of resources.

5 Conclusion and Future work

We proposed a failure handling method in SDN based on the best features of source routing and segment routing. Through simulations, we showed that proposed SSR method reduces memory overhead on the switches, computation time at the controller, and the packet size while providing almost the same performance of pure source routing in terms of the cost of the backup paths and the link utilization increase after the failure.

To demonstrate the potential benefits of the proposed SSR framework, we used simple heuristics in addressing various challenges in SSR. We now plan to

further investigate new algorithms to address these challenges. Both PSR and our SSR simply computes the shortest path as a backup path. We plan to develop new cost parameters that can take into account the interference of backup paths on the primary paths.

References

1. Antonio Capone, Carmelo Cascone, Alessandro Q. T. Nguyen, and Brunilde Sansò. Detour planning for fast and reliable failure recovery in SDN with openstate. *CoRR*, abs/1411.7711, 2014.
2. D. E. Comer. *Internetworking with TCP/IP*, volume I. Prentice Hall, Inc., third edition, 1995.
3. C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois. The segment routing architecture. In *IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, Dec 2015.
4. Klaus-Tycho Foerster, Stefan Schmid, and Stefano Vissicchio. Survey of consistent network updates. *CoRR*, abs/1609.02305, 2016.
5. K. T. Frster, R. Mahajan, and R. Wattenhofer. Consistent updates in software defined networks: On dependencies, loop freedom, and blackholes. In *IFIP Networking Conference (IFIP Networking) and Workshops*, pages 1–9, May 2016.
6. F. Hao, M. Kodialam, and T. V. Lakshman. Optimizing restoration with segment routing. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, April 2016.
7. N. Kitsuwan, D. B. Payne, and M. Ruffini. A novel protection design for openflow-based networks. In *The 16th International Conference on Transparent Optical Networks (ICTON)*, pages 1–5, July 2014.
8. Huang Liaoruo, Shen Qingguo, and Shao Wenjuan. A source routing based link protection method for link failure in sdn. In *The 2nd IEEE International Conference on Computer and Communications (ICCC)*, pages 2588–2594, Oct 2016.
9. 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 *IEEE International Conference on Communications (ICC)*, pages 1–6, May 2016.
10. R. M. Ramos, M. Martinello, and C. Esteve Rothenberg. Slickflow: Resilient source routing in data center networks unlocked by openflow. In *The 38th Annual IEEE Conference on Local Computer Networks*, pages 606–613, Oct 2013.
11. A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi. Openflow-based segment protection in ethernet networks. *IEEE/OSA Journal of Optical Communications and Networking*, 5(9):1066–1075, Sept 2013.
12. S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester. Enabling fast failure recovery in openflow networks. In *The 8th International Workshop on the Design of Reliable Communication Networks (DRCN)*, pages 164–171, Oct 2011.
13. Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet, and Piet Demeester. Openflow: Meeting carrier-grade recovery requirements. *Computer Communications*, 36(6):656 – 665, 2013. Reliable Network-based Services.
14. M. Soliman, B. Nandy, I. Lambadaris, and P. Ashwood-Smith. Exploring source routed forwarding in sdn-based wans. In *IEEE International Conference on Communications (ICC)*, pages 3070–3075, June 2014.