A Source Routing Based Link Protection Method for Link Failure in SDN

Huang Liaoruo, Shen Qingguo
Communication Engineering Institute
PLA University of Science and Technology
Nanjing, China
e-mail: huangliaoruo@sina.com; shenqg2@163.com

Shao Wenjuan
Zi Jin College
Nanjing University of Science and Technology
Nanjing, China
e-mail: Shaowj_nj@139.com

***Abstract*—The link failure recovery process may introduce large delay and serious packet loss in SDN because the failure has to be handled by remote controller. In recent years, several link protection methods have been put forward to solving this problem using preplanned backup paths, and avoiding the involvement of remote controller. However, it may need large quantities of flow entries to build these backup paths and complex mechanism to keep them active, and may cause congestion during rerouting process. In order to solve these problems, this paper proposes a source routing based link protection method in SDN. With predefined backup paths for every link, our method uses source routing to control the flow's route and steers the packet into backup path through updating the source routing header when link failure happens. Through implementation and evaluation, it is shown that this method can significantly reduce the overhead to build and maintain backup paths because we store the route information in the packet's header instead of flow entries installed in switch. And with flexibly and dynamically backup path planning, this method can achieve better congestion avoidance in the backup path during rerouting process comparing to previous methods.**

Keywords-source routing; link protection; link failure; SDN

I. INTRODUCTION

When link failure happens, switch has to depend on the remote controller to handle the failure and compute a new path for the interrupted flow because of the separation of control plan and data plan in SDN. Due to the delay between switch and controller, the failure recovery process may introduce heavy packet loss and thus cause service interruption. In order to solve this problem, predefined backup paths are used to reroute the interrupted flow avoiding the involvement of remote controller so that it can reduce the failover time in SDN. Path protection is the method which uses end-to-end backup paths and may cause packet loss of the flow on the fly. Therefore, an improved version of path protection called link protection or segment protection is put forward. It builds backup paths for every link along the working path so that the switch adjacent to the failure node can perform local link failure recovery and further reduce the failover time and packet lose when failure occurs.

Through experiments and simulations, it can be demonstrated that link protection method can significantly reduce the failure recovery time in SDN and meet the demand of carrier grade network. However, it also has some disadvantages: First, it needs large quantities of flow entries to build backup paths and complex mechanism to keep them active; second, the preplanned backup paths once constructed are hard to be modified or adjusted according to the network status so that it may cause congestion during rerouting process. So a dynamic and flexible link protection method is needed.

In order to solve the problems of link protection methods mentioned above, this paper proposes a source routing based link protection method which can significantly reduce the overhead to build backup paths and realize dynamically and flexibly backup path planning according to the network status. Our method uses source routing to control packet's behavior at the edge of network and steers packet into backup path through updating the packet's source routing header. With source routing SDN, we store the packet's route in the source routing header rather than the flow entries installed in switches, so that it can significantly reduce the number of entries to build paths. Furthermore, source routing also provides the ability to flexibly control the packet's behavior through programming the source routing header rather than the flow entries along the working path. Thus, with our method, we can easily modify, delete and reconstruct the packet's route without touching the switches and their flow entries. So it can provide flexibly and fast backup path constructing according to the network status and therefore avoid the congestion problem of backup path, which has been ignored by most of link failure recovery methods.

II. REALATED WORKS

Link failure recovery is one of the hot topics in SDN. There are two mechanisms to handle the link failure: reactive and proactive. The typical reactive method is fast path restoration proposed in [1]. With this method, the switch informs the controller of topology change event. The controller computes and installs the new path for the interrupted flows. Besides, authors of [1] also propose an improved version of fast path restoration called "predetermined restoration". Different from fast restoration, the backup paths are preplanned with priorities and controller can pick a proper backup path for the disrupted flows. The main short coming of reactive method is the failure handling process is depend on the remote controller and hard to meet the requirement of recovery time in carrier grade network.

978-1-4673-9026-2/16/$31.00 ©2016 IEEE

To overcome the disadvantages of reactive methods, proactive methods use static predefined backup paths to reroute the interrupted flows without involvement of the controller. And the key of the proactive methods is a handover mechanism which can disable the flow entries of old paths and then automatically put the backup path's flow entries into use. Before the Openflow v1.4, there are no such mechanisms but entry expiry timer as "idle time out" and "hard time out". So in order to realize fast proactive link failure, [2] proposes a method called "Auto-Reject" which can detect the link failure and disable the flow entries. In [3], authors put forward a flow entry expiry mechanism to automatically delete the entries of old path when link failure occurs. In [4], authors use Openstate to monitor the packet and switch ports state and reallocate flows to new paths when switch port is down. Openflow v1.4 provides a fast failover mechanism itself through instantiating multi action list for the same flow entry and applying them according to the link status [5].

Based on these handover mechanisms above, several proactive methods have been put forward. In order to minimize the failure recovery time, the authors of [2] propose an improvement of path protection called segment protection. In this scheme, backup paths are planned on every hop instead of end-to-end, so that flow can be rerouted at local switch adjacent to the fail link and achieve shorter failure recovery time. However, segment protection needs large quantities of flow entries to build these paths and complex mechanism to keep them active.

In order to reduce the number of flow entry to build backup paths, authors of [6] put forward a novel design of SDN network. They abstract network into two planes: working plane and transient plane. The flow is transmitted through working plane when network is normal. And transient plane is consisted of links which can be shared among different backup paths so that the number of flow entries to build backup paths is reduced. When link failure happens, packet can be handed to the transient plan through inserting a tag into the packet header. But it still needs many flow entries to build transient plane and still hard to avoid congestion during rerouting process.

The authors of [7] propose a backup path optimization method which put the link utilization into consideration. Flows from different input links are split among different backup paths so than it can avoid the congestion in backup paths. Although using the optimized backup paths can reduce the possibility of congestion, but it needs too many flow entries to build multi backup paths for every link and still lacks of flexibility as the backup paths are static planned.

In [4], authors propose a multi path rerouting method based on Openstate. This method improves the segment protection with planning multi-backup paths at every hop and splitting flows among these paths to solve the congestion problem. With an optimization model, this method can significantly reduce the possibility of congestion, but is still static planned and need too many flow entries to build backup paths.

A source routing based method called SlickFlow is proposed in [8]. It encodes all the backup paths into the packet header. When link failure occurs, switch looks up the backup path and reroute the packet. Obviously, this method may introduce large overhead to the packet header.

In summary, there is no link protection method that can provide fast failure recovery ability and flexibly backup path planning ability with a low overhead for now. And the main reason of that is the tight bound of flow and the path it transmits in SDN. In our method, we use source routing to separate the flow and its path so that it can achieve congestion aware fast link failure with a very low overhead.

III. SOURCE ROUTING BASED SDN

A. *Packet Header Encoding*

Recent years, source routing SDN has attracted a lot of attention as it can reduce the number of entries used in SDN without losing the programmability of flows at the same time [9]-[11]. However, there is no source routing mechanism in the latest Openflow, so in this paper we modify the VLAN tag of 802.1q to realize source routing. The vlan_id field is used to encode route information.

In source routing SDN, the next hop of packet is presented as output port ID which is encoded in vlan_id(VID) field as figure 1 shows:

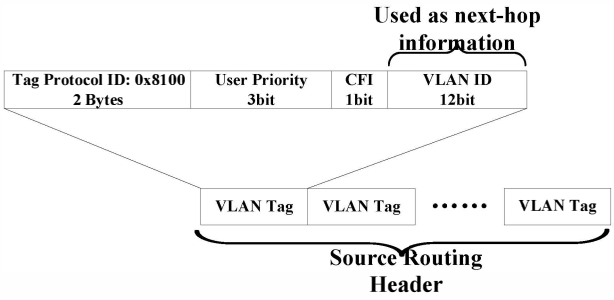

Figure 1. Packet header encoding.

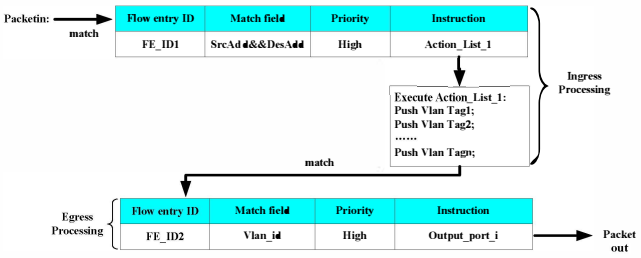

Figure 2. Packet header programming.

The length of VID field in VLAN tag of 802.1q is 12bit so that it can support 2^{12} of next hops at most. As figure 1 shows, to form the complete route information, it has to nest several VLAN tags into the packet's header. Then switch outputs the packet through matching the VID field of the outer most VLAN tag and deletes the outer most tag after matching.

B. *Source Routing Based SDN*

Source routing SDN mainly has three functions: packet header programming, packet forwarding and packet steering.

1) Packet header programming

Packet header programming assembles the packet's source routing header with VLAN tags introduced in 3.1 according to the flow entries and action-list preinstalled by controller as figure 2 shows:

With the Push_vlan_tag action provided by Openflow, switch can nest several VLAN tags to the header of the packet so that the route information is formed.

2) Packet forwarding

Packet forwarding process forwards the packet according to the next hop information formed in packet header programming process as figure 3 shows:

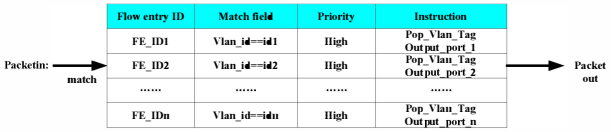

Figure 3. Packet forwarding.

Through matching the VID field, switch finds the proper port to output the packet and delete the outer most VLAN tag in packet's header with Pop_Vlan_Tag action provided by Openflow.

3) Packet steering

Packet steering steers the packet to a new path through updating the source routing header as figure 4 shows:

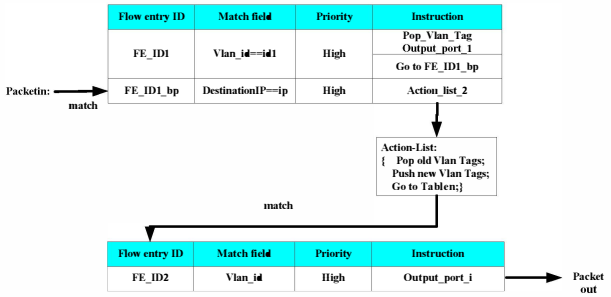

Figure 4. Packet steering process.

When link failure occurs, packet is sent to the fast failover group table and switch modifies the status of the original port according to the failure detection result. Then similar to the packet header programming process, switch update the old packet header with a new one through pop and push VLAN tags according to the predefined action list so that packet can be transmitted along a new path.

IV. SOURCE ROUTING BASED LINK PROTECTION

A. Procedures of the Method

There are two procedures in the proposed method: backup path planning and fast failure recovery.

1) Backup path planning

In the backup path planning procedure, controller computes and builds backup paths for every link of the working path according to the network status information gathered from switches and other devices as figure 5 shows:

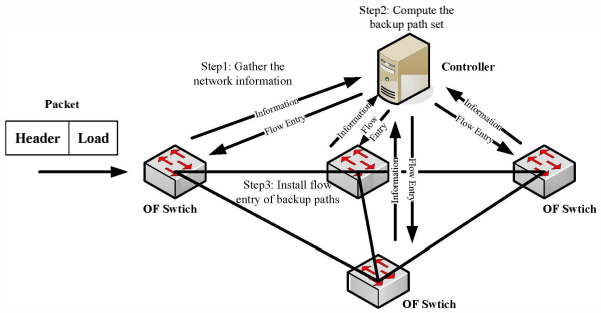

Figure 5. Backup path planning procedure.

The function of the flow entries which are installed by controller in the backup path planning procedure is updating the packet's source routing header. The backup path planning procedure is performed to find the best backup path according to the network status so as to avoid congestion when rerouting the interrupted flows into backup path. The principles to build these backup paths are introduced in 4.3.

2) Fast failure recovery

The fast failure recovery procedure is performed at the switch adjacent to the failure node without the involvement of remote controller so that it can minimize the time of fast failover. The architecture of this procedure is shown in figure 6

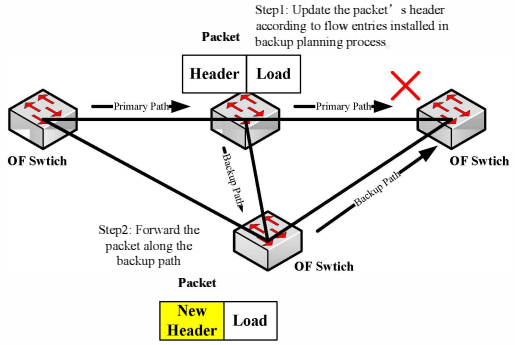

Figure 6. Fast failure recovery procedure.

With flow entries installed in backup path planning procedure, switch adjacent to the failure node can change the packet's header through popping and pushing a sequence of VLAN headers and then transmit the modified packet through the backup path. Therefore, the local fast failure recovery is performed and the packet can bypass the failure link.

B. Examples

In this section, we carry out an example to explain our method. Consider a network like figure 7.

The working path is $S \rightarrow D \rightarrow T$ which is from source node S to destination node T. At the backup path planning procedure, controller build backup path $S \rightarrow A \rightarrow B \rightarrow C \rightarrow T$ for

link $S \rightarrow D$ and backup path $D \rightarrow B \rightarrow C \rightarrow T$ for link $D \rightarrow T$. When network is normal, the source routing header of packet can be presented as "11" for the simplicity, which is the sequence of the output ports. We presume that link failure happens at link $D \rightarrow T$.

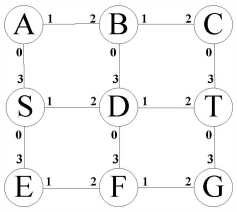

Figure 7. Network topology.

At node S, the source routing header of the packet is formed as "11". Then S checks this source routing header and outputs the packet through port 1. At the same time, it deletes the outermost VLAN tag which is "1" and the packet's source routing header changes to "1".

At node D, the source routing header of the packet is "1". Then D checks this header and finds out that port1 is down so that it hands the packet to the fast failover group which is installed at backup path planning procedure. According to these flow entries, the packet's source routing header changes from "1" to "310" which represents the backup path $D \rightarrow B \rightarrow C \rightarrow T$ through popping and pushing VLAN tags. At last, the packet is checked again and transmitted through port3.

C. Backup Path Planning

Let's defined the following parameters:

B	The set of backup paths which are selected;
F	The set of flows need to reroute;
D_{sd}	The set of paths from s to d;
e_{ij}^{p}	Indication function which is equal to 1 if link (i,j) belongs to path p, and 0 otherwise;
t_p^{B}	Indication function which is equal to 1 if path p is selected in B, and 0 otherwise;
c_{ij}	The available bandwidth of link (i,j);
o	The fail point o;
f_i	The flow i;
$c(f_i)$	The bandwidth demand of flow f_i;

In some link protection methods, the backup paths are shortest path from the failure recovery node to the destination node. However, the available bandwidth provided by backup path may not sufficient to reroute the interrupted flows so that causes congestion in the backup path. This is the congestion problem of the link failure recovery which has been ignored by most of the link protection methods. In order to solve this problem, we put the available bandwidth into consideration when planning the backup paths. We introduce the parameter γ ($\gamma \geqslant 1$) and lower bound of the available bandwidth provided by backup path as $\gamma \sum_i c(f_i)$. Then we formalized the backup path planning problem as a linear integer programming problem as follows:

$$\min \sum_p e_{ij}^p t_p^B$$

$$s.t \quad \min\{e_{ij}^p t_p^B c_{ij}\} \geq \gamma \sum_i c(f_i) \tag{1}$$

$$\sum_i e_{oi}^p = 0, \forall p \in D_{sd}, \forall i \in N \tag{3}$$

$$\sum_{(i,j)} e_{ij}^p \leq 1, \forall p \in D_{sd}, \forall (i,j) \in E \tag{4}$$

$$\sum_i e_{ij}^p \leq 2, \forall i \in N, \forall p \in D_{sd}, \forall (i,j) \in E \tag{5}$$

The target function demonstrates that we want to find a shortest path from the recovery node to the destination node. And this path should satisfy the lower bound of the available bandwidth which is represented by formula (1). Also, the backup path should not contain the failure node which is guaranteed by formula (2). Formula (4) and (5) represent that backup path is loop free.

To solve the problem above, we propose a modified Dijkstra algorithm with two procedures. The first procedure of the algorithm deletes the link which can't provide sufficient available bandwidth as line 1 shows. The second procedure of the algorithm finds the shortest path as normal Dijkstra algorithm does, which is shown in line 2~17.

```
Algorithm ModifiedDijkstra(graph, start, destin, bandW)
 1:  graph = deleteLinks(graph,bandW)
 2:  for i = 0 to number of nodes do
 3:    pathList.add(start)
 4:    pathList.add(i)
 5:    pathListMap.put(i,pathList)
 6:  end for
 7:  for bridge = 0 to number of nodes do
 8:    for next = 0 to number of nodes do
 9:      if
       startTo(bridge)+getLength(bridge,next)<startTo(next)
       then
10:       path = pathListMap.get(next)
11:       bridgePath = pathListMap.get(bridge)
12:       path.clear()
13:       path.addAll(bridgePath)
14:       path.add(next)
15:     end if
16:   end for
17: end for
```

V. Implementation and Evaluation

A. Implementation

In order to verify the efficiency of our method, we build our method on an online test bed supported by SDNLAB.

The test bed is consisted of a software controller installed in Ubuntu and software openflow-enabled switches. The controller used is OpendayLight Desktop Litmus and the switch used is Open vSwtich. We realize the source routing SDN and carry out experiments on failure recovery time.

For the sake of demonstration, we compare our method with path protection and segment protection. In the path protection, the recovery time consists of three parts as (6) shows:

$$T_{recovery} = T_{detection} + T_{propaganda} + \sum_{f}^{'} T_{handover,f} \quad (6)$$

The $T_{detection}$ is the failure detection time and the $T_{propaganda}$ is the time for informing the source node of the link failure. $T_{handover,f}$ is the time to activate the backup path of flow f in switch. In the segment protection, the recovery time consists of two parts as (7) shows:

$$T_{recovery} = T_{detection} + \sum_{f}^{'} T_{handover,f} \quad (7)$$

In the method we propose, the recovery time can be presented as (8):

$$T_{recovery} = T_{detection} + \sum_{f}^{'} T_{update,f} \quad (8)$$

The $T_{update,f}$ is the time to activate the backup path and modify the source routing header of flow f in switch. The $T_{update,f}$ may larger than the $T_{handover}$ as the former has to perform additional actions to modify the packet's header. However, in the open vSwtich we carry out evaluation, the difference is very small even the packet number is very large. In our experiments, we use out-of-band control mode and four matrix topologies with different scale to evaluate the recovery time of three methods. We use in-net failure detection method introduced by [12] which can achieve a detection time below 5ms. And the result is shown in figure 8.

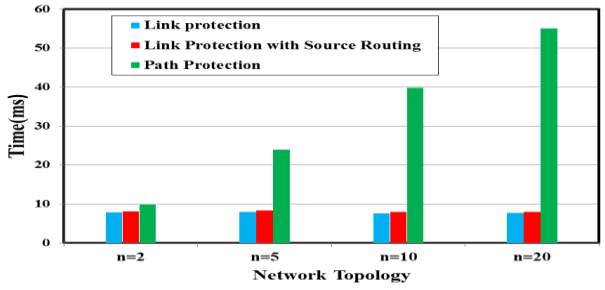

Figure 8. Total recovery time in different topologies.

Through the evaluation, the time to activate backup path in switch is 5.8ms in average. And it can be seen from the figure 8 that due to the larger propaganda time with the expanding of network, failure recovery time is increasing in path protection method. However, in other two methods, the link recovery time is almost the same because the time consumption of the link recovery is independent of the network scale. It can be concluded that our method can achieve the same time consumption as the segment protection and other methods that perform local flow rerouting, which is much less than 50ms.

B. Flow Entry Consumption

In this section, we evaluate the flow entry consumption of our method in the worst case and compare our method with segment protection and the ITP method proposed in [6].

Consider an n*n matrix network like figure 9 shows.

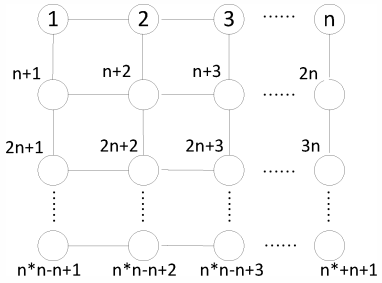

Figure 9. n*n matrix network.

It has been demonstrated in [2] and [6] that the flow entries of segment protection can be presented as (9):

$$F_{segment\_protection} = \sum_{i=2}^{n}\sum_{j=1}^{i}(N_{i,j} \times M_{n,i,j}) \quad (9)$$

And $N_{i,j}$, $M_{n,i,j}$can be expressed by (10) and (11):

$$N_{i,j} = \begin{cases} 2j+4i-4; j<3 \\ 4i+4j-8; j \geq 3 \end{cases} \quad (10)$$

$$M_{n,i,j} = (n-i+1)\times(n-j+1)\times\sigma_{i,j}; \sigma_{i,j} = \begin{cases} 8, if\ j \neq 1, i \neq j \\ 4, if\ i=1\ or\ i=j \end{cases} \quad (11)$$

In independent transient plane (ITP) method, the total number of flow entries can be presented as (12)~(14):

$$F_{ITP} = F_{working\_path} + F_{transient\_plane} \quad (12)$$

$$F_{transient\_plane} = n^2 \times [2n^2 + n - 2] \quad (13)$$

$$F_{working\_path} = \sum_{l=2}^{n}\sum_{j=1}^{i}[(i+j-1)\times M_{n,i,j}] \quad (14)$$

In the method we propose, the total flow entries can be expressed as (15):

$$F_{SR-SDN} = F_{working\_path} + F_{transient\_plane} \quad (15)$$

In the source routing schema, flow entries that find right port to output packet can be shared by all working paths. So the flow entries needed to build working paths can be presented as (16):

$$F_{working\_path} = n^2 \times p \tag{16}$$

p is the number of ports in every switch. As discussed in III, we construct one backup path for every link and it needs one flow entry to update the packet's header at every hop. So the flow entries to build backup paths can be presented as (17):

$$F_{backup\_path} = \sum_{i=2}^{n} \sum_{j=1}^{i} [(i+j-1) \times M_{n,i,j}] \tag{17}$$

So the total flow entries of three methods is shown in figure 10.

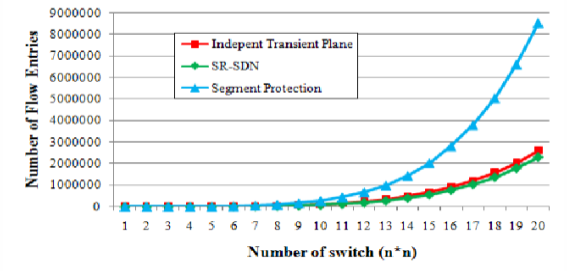

Figure 10. Total flow entries in n*n network.

It can be seen that our method has the minimal flow entry consumption among three methods and the advantage is expanding with the incensement of network scale. And the improvement of ITP and SR-SDN compared to segment protection is shown in figure 11. It can be demonstrated that our method can realize less flow entry overhead compared to ITP.

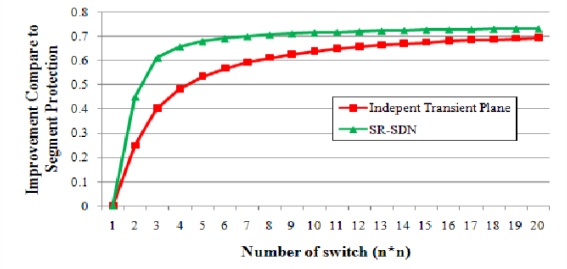

Figure 11. Improvement of flow entries compared to segment protection.

C. *Dynamic Backup Path Planning Evaluation*

To evaluate the benefit of dynamic backup path planning, we derive simulations on OMNET++ with INET. We use 5*5 matrix network and SDN structure introduced in [13]. Every edge node of the network connects to a host which randomly sent packets to another. And the interval of packet arrival follows exponential distribution with λ =0.2. The backup path planning process is conducted every 0.1s. The max transmit rate of switch is 1Mbps. We execute simulations of SR-SDN and segment protection, and record the link utilization of backup path during the simulation. The average link utilization of backup path during rerouting process can be reduced as 35.2% which is shown in figure 12.

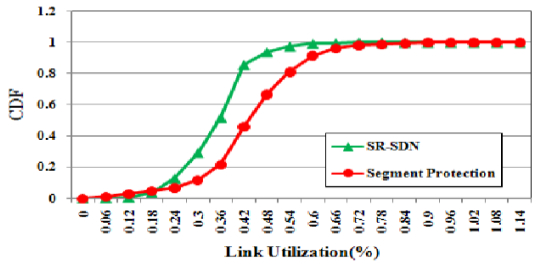

Figure 12. The CDF of backup path's link utilization.

From the Figure 12, it can be seen that comparing to the static planned backup path, dynamically planned backup path according to network status using source routing can significantly reduce the possibility of congestion during rerouting process. At the same time, with the improvement of the link utilization, the delay of the rerouted packet is also decreased and the result is shown in Figure 13.

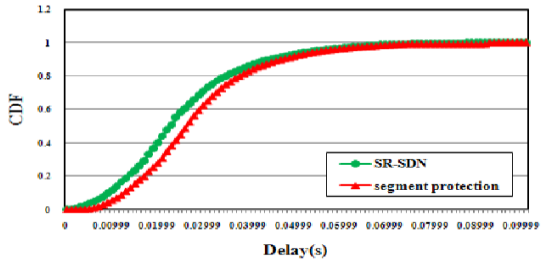

Figure 13. The CDF of the packets' delay.

As figure 13 shows, through choosing the less congestion backup path, the average delay of rerouted packet can be reduced 12.1% compared to segment protection.

In summary, comparing to the exists SDN link protection methods, the link protection method based on source routing SDN proposed by us can significantly reduce the flow entry consumption and realize flexible and dynamical backup path planning so as to avoid congestion problem during rerouting process.

VI. CONCLUSIONS

In this paper, we propose a source routing based link protection method for link failure in SDN. Compared with other failure recovery methods used in SDN, it mainly has three advantages: First, it can perform local link failure recovery so that the time consumption is much less than 50ms; Second, it can realize fast link failure recovery with a very low flow entry consumption; Third, it can easily realize flexible and dynamic backup path planning according to network status. Finally, through evaluations and simulations,

it can be concluded that the method proposed in this paper can achieve better performance than the latest failure recovery methods. In the next step research, we will address the detail issues during the implementation process and explore the application of source routing SDN in other fields.

ACKNOWLEDGMENT

This work is financially supported by the Natural Science Foundation of China (61271254).

REFERENCES

[1] S. Sharma, D. Staessens, D. Colle, M. Pickavet and P. Demeester, "Enabling fast failure recovery in OpenFlow networks," *Design of Reliable Communication Networks (DRCN), 2011 8th International Workshop on the*, Krakow, 2011, pp. 164-171.doi: 10.1109/DRCN.2011.6076899

[2] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci and P. Castoldi, "OpenFlow-based segment protection in Ethernet networks," in *IEEE/OSA Journal of Optical Communications and Networking*, vol. 5, no. 9, pp. 1066-1075, Sept. 2013.doi: 10.1364/JOCN.5.001066

[3] Yang Yu, Li Xin, Chen Shanzhi and Wang Yan, "A framework of using OpenFlow to handle transient link failure," *Transportation, Mechanical, and Electrical Engineering (TMEE), 2011 International Conference on*, Changchun, 2011, pp. 2050-2053.doi: 10.1109/TMEE.2011.6199619

[4] A. Capone, C. Cascone, A. Q. T. Nguyen and B. Sansò, "Detour planning for fast and reliable failure recovery in SDN with OpenState," *Design of Reliable Communication Networks (DRCN), 2015 11th International Conference on the*, Kansas City, MO, 2015, pp. 25-32.doi: 10.1109/DRCN.2015.7148981.

[5] Pankaj Thorat, et al. "Optimized self-healing framework for software defined networks," *Ubiquitous IInformation Management and Communication(IMCOM)*, *2015 9th International Conference on,* BALI, 2015, pp. 1-6. doi:10.1145/2701126.2701235P.

[6] N. Kitsuwan, D. B. Payne and M. Ruffini, "A novel protection design for OpenFlow-based networks," *2014 16th International Conference on Transparent Optical Networks (ICTON)*, Graz, 2014, pp. 1-5. doi: 10.1109/ICTON.2014.6876515.

[7] S. S. W. Lee, K. Y. Li, K. Y. Chan, G. H. Lai and Y. C. Chung, "Path layout planning and software based fast failure detection in survivable OpenFlow networks," *Design of Reliable Communication Networks (DRCN), 2014 10th International Conference on the,* Ghent, 2014, pp. 1-8.doi: 10.1109/DRCN.2014.6816141.

[8] R. M. Ramos, M. Martinello and C. Esteve Rothenberg, "SlickFlow: Resilient source routing in Data Center Networks unlocked by OpenFlow," *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, Sydney, NSW, 2013, pp. 606-613.doi: 10.1109/LCN.2013.6761297.

[9] M. Soliman, B. Nandy, I. Lambadaris and P. Ashwood-Smith, "Exploring source routed forwarding in SDN-based WANs," *2014 IEEE International Conference on Communications (ICC),* Sydney, NSW, 2014, pp. 3070-3075.doi: 10.1109/ICC.2014.6883792.

[10] Soliman, M., Nandy, B., Lambadaris, I., & Ashwood-Smith, P. "Source routed forwarding with software defined control, considerations and implications," 2012 *ACM Conference on CONEXT Student Workshop*, Nice, France,2012, pp.43-44. doi:10.1145/2413247.2413274.

[11] Jyothi, Sangeetha Abdu, M. Dong, and P. B. Godfrey. "Towards a flexible data center fabric with source routing,"*2015 1st ACM SIGCOMM Symposium on Software Defined Networking Research,* NewYork, USA, 2015, pp.1-8. doi:10.1145/2774993.2775005.

[12] N. L. M. v. Adrichem, B. J. v. Asten and F. A. Kuipers, "Fast Recovery in Software-Defined Networks," *2014 Third European Workshop on Software Defined Networks*, Budapest, 2014, pp. 61-66.doi: 10.1109/EWSDN.2014.13.

[13] Klein, Dominik, and M. Jarschel, "An OpenFlow extension for the OMNeT++ INET framework." *International ICST Conference on Simulation TOOLS and Techniques* 2013, pp. 322-329. doi:10.4108/icst.simutools.2013.251722.