Congestion Aware Fast Link Failure Recovery of SDN Network Based on Source Routing

Liaoruo Huang[1], Qingguo Shen[1] and Wenjuan Shao[1,2]
[1] College of Communications Engineering, PLA University of Science and Technology
Nanjing, Jiangsu 210007 - China
[e-mail: huangliaoruo@**sina**.com, shenqg2016@163.com]
[2] Zijin College, Nanjing University of Science and Technology
Nanjing, Jiangsu 210094 - China
[e-mail: shaowj_nj@139.com]
*Corresponding author: Qingguo Shen

Received May 19, 2016; revised September 30, 2016; accepted July 17, 2017; published November 30, 2017

Abstract

The separation of control plane and data plane in Software Defined Network (SDN) makes it flexible to control the network behavior, while also causes some inconveniences to the link failure recovery due to the delay between fail point and the controller. To avoid delay and packet loss, pre-defined backup paths are used to reroute the disrupted flows when failure occurs. However, it may introduce large overhead to build and maintain these backup paths and is hard to dynamically construct backup paths according to the network status so as to avoid congestion during rerouting process. In order to realize congestion aware fast link failure recovery, this paper proposes a novel method which installs multi backup paths for every link via source routing and per-hop-tags and spread flows into different paths at fail point to avoid congestion. We carry out experiments and simulations to evaluate the performance of the method and the results demonstrate that our method can achieve congestion aware fast link failure recovery in SDN with a very low overhead.

Keywords: Software-Defined Network; link failure recovery; link protection; source routing;

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

https://doi.org/10.3837/tiis.2017.11.002 **ISSN : 1976-7277**

1. Introduction

The separation of data plane and control plane in Software Defined Network (SDN) provides programming capabilities for network and flow transmission, bringing convenience to the network virtualization, load balancing, QoS and other applications. However, this separation also makes it difficult to perform fast link failure recovery in SDN, because the network depends on the remote controller to handle the failure, which may introduce large delay, therefore cause serious packet loss and interruption of network service.

In order to satisfy the requirement of recovery time in carrier grade network, several proactive mechanisms have been put forward to realize approximate zero loss and fast link failure recovery in SDN[1]~[5]. In proactive mechanisms, end-to-end or per-hop preplanned backup paths are used to guarantee that when link failure occurs, the switch adjacent to the fail point can reroute the flow immediately through the backup paths to the destination without the controller's decision. Through experiments, it is shown that they can achieve much shorter recovery time than the traditional methods. However, there are still some concerns about the proactive mechanism: First, it may need large quantities of flow entries to build backup paths and complex mechanism to keep them active; second, the preplanned backup paths once constructed, are inconvenient to be modified, deleted and rebuilt, so it is hard to adaptively adjust backup paths according to network status and therefore may cause congestion when rerouting the interrupted flows.

In summary, there are three objectives that failure recovery method has to achieve: (1)Fast: for the requirements of critical tasks in carrier grade network, it has to be as fast as possible to detect and locate the failure point and react to the link failure. And the total failure recovery time should be less than 50ms[6]; (2) Low overhead: the flow entries that backup paths consume should be as few as possible so that the network can accept more flows; (3) Congestion aware: the backup path planning and flow rerouting process should put the network status into consideration, so as to avoid congestion of other parts of network.

There has been a lot of works concentrating on the objective (1) and (2)[1]~[7], but few works notice the congestion problem during rerouting process. In the traditional way, it has to reserve bandwidth for the protected flows on the backup paths to avoid packet loss and service degradation caused by congestion[8]. However, this method may lead to waste of network resource and is complex to deploy in practice. In order to achieve the three objectives above, this paper proposes a congestion aware fast link failure recovery method of SDN. The method uses source routing to control flow's route at the edge of the network. According to the predefined multi backup paths for every link and the bandwidth demand of every flow, the controller inserts per-hop backup path tag to the source routing header of the packet to identify the specific backup path for the flow at every hop. With this mechanism, the interrupted flows are transmitted through multiple backup paths during rerouting process so that the congestion can be avoided. When failure occurs, the switch adjacent to the fail point updates the source routing header of the packet according to the backup path tag, then the packet can bypass the fail point. This method takes advantages of topology independent feature of source routing and decouples the backup path planning and backup path selection. Therefore, it can realize dynamically flow assignment among backup paths according to the network status.

Compared with the traditional methods, our method mainly has four advantages: First, it can perform local link failure recovery without involvement of the controller so that it can minimize the recovery time. Second, it can achieve rather low flow entry consumption

because it stores route information in the packet header rather than flow entries installed in the switch. Third, it considers the congestion problem during rerouting process which is ignored by most of the traditional methods, and can significantly reduce the possibility of congestion. Fourth, it can be applied to handle both single link failure and single node failure. The main contributions of this paper are following: (1) We propose a novel architecture and implementation of SDN based on source routing; (2) We propose a congestion aware fast link failure recovery method based on source routing SDN; (3) We give the formalization and solution to the backup path planning and selection problems; (4) We analyze the overhead and evaluate the performance of this method.

The remainder of this paper is organized as follows: Section 2 gives some relate works of SDN link failure recovery; Section 3 introduces the architecture of the proposed method. Section 4 gives formalization and solution to the backup path planning and selection problems; Section 5 gives the implementation of proposed method; Section 6 carries out evaluation and simulations of the proposed method; Section 7 gives the conclusion.

2. Related Work

2.1 Link Failure Detection

The link failure recovery time is consisted of two parts: failure detection time and reaction time. In order to realize fast link failure recovery, it is important to detect and locate the failure immediately after the failure happens. Loss of Signal (LOS) and Bidirectional Forwarding Detection (BFD)[9] are widely used to detect link failures. However, the time of failure detection using LOS may be hundreds of milliseconds which can't meet the requirement of carrier grade network. BFD uses a simple "hello-echo" protocol between two end terminals of a path and the time consumption is below 50ms[10]. But, BFD is unable to locate the failure link in the path so that it can be only used to detect failure of a path rather than a specific link. To solve this problem, authors of [11] propose a per-link BFD mechanism which applies the BFD between the terminals of a link rather than a path. Through experiment, it can complete failure detection within 10ms.

Besides, in [12] and [13], an path monitoring and failure location mechanism is proposed. With a monitoring cycle covering all links in a given network, the Link Status Monitoring (LSM) packet is used to supervise the path alive status. The switches which have receive LSM packet send the Failure Location Identification (FLI) packets to the controller. So when there is a link failure, the failure node can be precisely located according to the FLI packets. Moreover, authors of **[14]** put forward a link failure detection mechanism which is applied to detect the failure of control channel between switches and the controller using the information sharing mechanism between multiple controllers.

With the efficient and fast detection methods above, our failure recovery method can be realized and the focus of our paper is how to minimize the reaction time after failure detection.

2.2 Reactive Link Failure Recovery

There are two mechanisms to handle the link failure: reactive and proactive. The typical reactive method is fast path restoration proposed in [15], [16] and [17]. With this method, the switch informs the controller of topology change event. Then the controller computes and installs the new path for the interrupted flows. The time consumption of this process may be large, therefore it may cause packet loss and service interruption. Besides, authors of [15] also propose an improved version of fast path restoration called "predetermined restoration".

Different from fast restoration, the backup paths are preplanned with priorities and the controller can pick a proper backup path for the disrupted flows.

The reactive methods mainly have two disadvantages: First, due to the involvement of the remote controller, the time consumption of the link failure recovery can't meet the demand of carrier grade network, which is demonstrated through experiments in [10] and [18]; Second, computing and installing paths for interrupted flows may introduce heavy burden to the controller, and therefore degrade the performance of the entire network.

2.3 Proactive Link Failure Recovery

To overcome the disadvantages of reactive methods, proactive methods use static preplanned 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. In Openflow-enabled switches, there is no such mechanism but entry expiry timer as "idle time out" and "hard time out". So in order to realize fast proactive link failure, [1] proposes a method called "Auto-Reject" which can detect the link failure and disable the flow entries. In [19], authors put forward a flow entry expiry mechanism to automatically delete the entries of old path when link failure occurs. In [2], authors use Openstate to monitor the packet and switch ports state, and reallocate flow to new path when switch port is down. However, the mechanisms mentioned above need additional modification to the switch and are hard to be applied in practice. The Openflow v1.5 provides a fast failover mechanism called failover group table through instantiating multi action lists for the same flow entry and applying them according to the link status[20]. It doesn't need any modification to the Openflow protocol and can achieve automatically handover according to the alive status of action lists.

Based on these handover mechanisms above, several proactive methods have been put forward. In order to minimize the failure recovery time, the authors of [1] propose an improvement of link 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 backup paths and complex mechanism to keep them active.

For the purpose of reducing the number of flow entries to build backup paths in segment protection, authors of [3] put forward a novel design called Independent Transient Plane (ITP) which consists of links that can be shared by different backup paths to reroute flows so that the number of flow entries is reduced. Besides, authors of [7] apply Loop-Free Alternates[21] to the Openflow-based network. With additional loop detection mechanism, it can achieve maximum protection coverage with a very low overhead. However, there are still two concerns about these methods: first, the consumption of flow entry is still large and need to be further optimized; second, the backup path planning process hasn't put link status into consideration so that it may cause congestion during rerouting process.

To solve the congestion problem, [13] proposes a backup path optimization method which put the link utilization into consideration. Optimization model is put forward and evaluated. Although using the optimized backup paths can reduce the possibility of congestion, but it is still not efficient enough because static paths can't be adaptively adjusted according to the network status.

In [2], authors propose a Multi Path Rerouting (MPR) method based on Openstate. This method improves the segment protection with planning multi-backup paths for every link and distributing flows into backup paths to solve the congestion problem. With an optimization

model, this method can significantly reduce the possibility of congestion, but the backup paths are still static planned and need too many flow entries.

A source routing based method called SlickFlow is proposed in [4]. It encodes all the backup paths into the packet header. When link failure occurs, switch looks up the backup path and reroutes the packet. Obviously, this method introduces too much overhead to the packet header and uses static backup paths as well.

In [22] and [23], the congestion aware fast link failure recovery method in IP and Openflow hybrid network is researched. Multi backup paths are pre-planned and installed in the Openflow switch. The IP switches forward interrupted packets directly to the Openflow switch when failure happens. Then Openflow switch picks a proper backup path to reroute the packet. Although this method can handle the congestion problem well, but it is efficient in IP network rather than SDN.

In summary, there is no method that can fulfill the three requirements mentioned in Section 1 at the same time for now, and the main reason of that is the tight coupling of flow and the path flow 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.

3. The Proposed Solution for Fast Link Failure Recovery

3.1 Packet Header Encoding

Recent years, source routing SDN has attracted some attention as it can reduce the number of entries used in SDN without losing the programmability of flows at the same time[24][25][26]. However, source routing is not yet supported in the latest Openflow. So in this paper we modify the VLAN tag of 802.1q to realize source routing. The vlan_id field and vlan_user_priority field are used to encode route information.

Different from traditional source routing, the next hop information has two parts in our method: output port ID and backup path tag. The output port ID is encoded in vlan_id(VID) field while backup path tag is encoded in vlan_user_priority(VUP) field as **Fig. 1** shows:

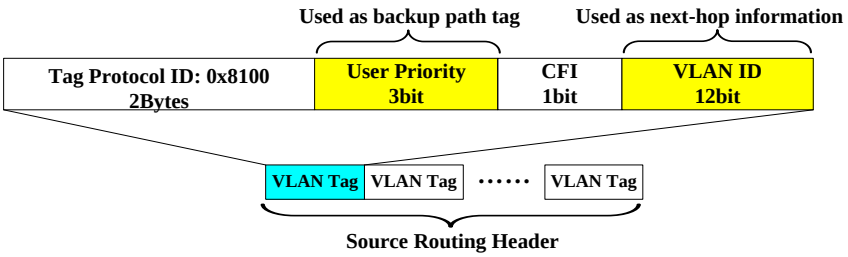


Fig. 1. Packet Header Encoding

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. And 8 backup paths can be supported at most as VUP field is 3bit long. The switch can transmit the packet along the working path through matching the VID field when network is normal, and steer the packet to backup path through matching the VUP field when link failure occurs.

In addition, the length of the VLAN tag and source routing header may introduce overhead to the packet and therefore limit the scalability of the source routing SDN. However, the length of packet header can be further optimized as there are only a few bits being used in VLAN tag.

We believe that with the arbitrary matching ability provided by Openflow, source routing mechanism with shorter length of packet header can be developed in the near future.

3.2 Source Routing SDN

Source Routing SDN (SR-SDN) mainly has three functions: packet header programming, packet forwarding and packet steering. In order to realize fast failure recovery, we leverage the fast failover group table of Openflow, which can perform different action list according to the specific situation.

(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 the controller as **Fig. 2** shows:

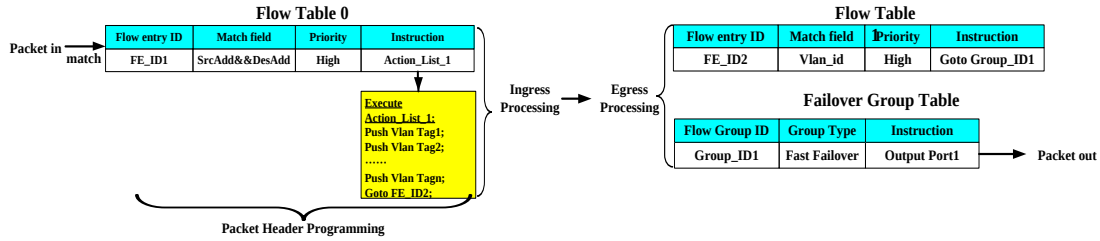


Fig. 2. Packet header programming

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 by packet header programming process as **Fig. 3** shows:

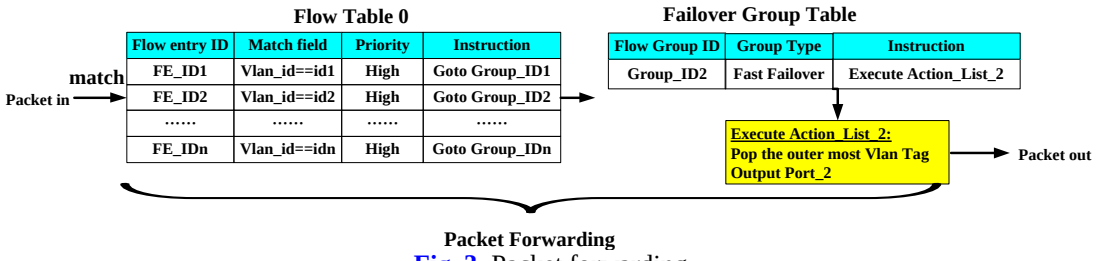


Fig. 3. Packet forwarding

Through matching the VID field, switch finds the proper port to output the packet and deletes 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 **Fig. 4** shows:

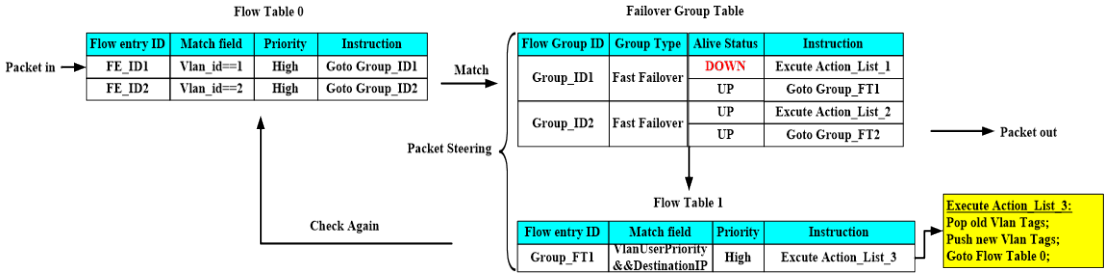


Fig.4 Packet steering process

When link failure occurs, the alive status of the primary action list in failover group table is changed to DOWN. Then the packet is processed by the secondary action list which updates the packet's source routing header through popping and pushing VLAN tags according to the backup path tag and the destination IP address. After updating process, the packet is checked again and output according to the new source routing header.

3.3 Procedures of the Method

The congestion aware fast link failure recovery method proposed in this paper contains 3 procedures: backup path planning, backup path selection, fast link failure recovery. The former two procedures are conducted in the controller and edge switches, and the last procedure is conducted in the core switches only when link failure occurs.

In the backup path planning process, according to the network status, the controller finds the optimal backup path set which can provide sufficient available bandwidth with minimal hops. Besides, the number of the backup paths for every link should be restricted to reduce the complexity of backup path planning algorithm and flow entry consumption for path construction. Then the controller tags every backup path and installs corresponding flow tables to the switches. The planning process is shown in **Fig. 5**:

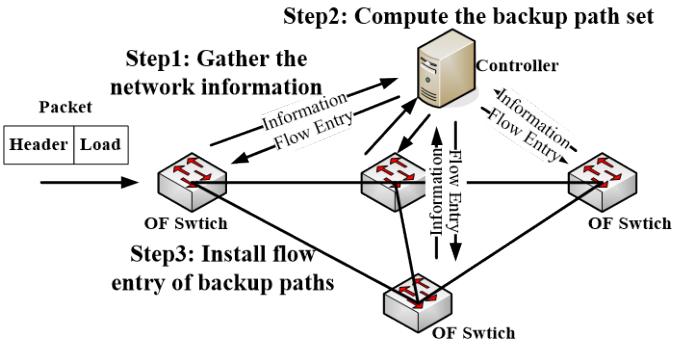


Fig. .5 The backup path planning process

In the backup paths selection process, according to the bandwidth demands of flows, the controller allocates every flow to a proper backup path at every hop and inserts the tag of selected backup path into the per hop information of packet header. The selection process is shown in **Fig. 6**:

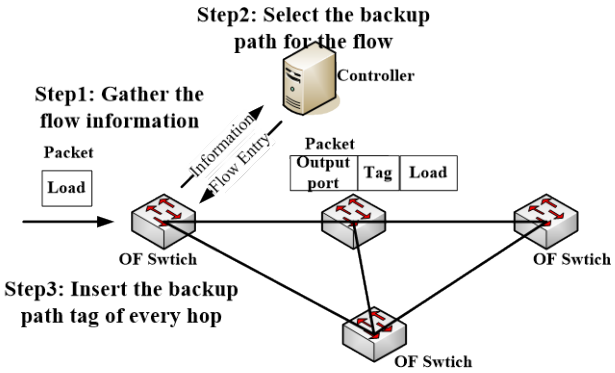


Fig. 6. The backup paths selection process

The fast link failure is performed only when link failure occurs. In fast link failure revocery process, the packets interrupted are processed by the fast failover group table, which can steer the packet to the proper backup path according to the backup path tag in the packet header and destination IP address. The failure recovery process is shown in **Fig. 7** as follows:

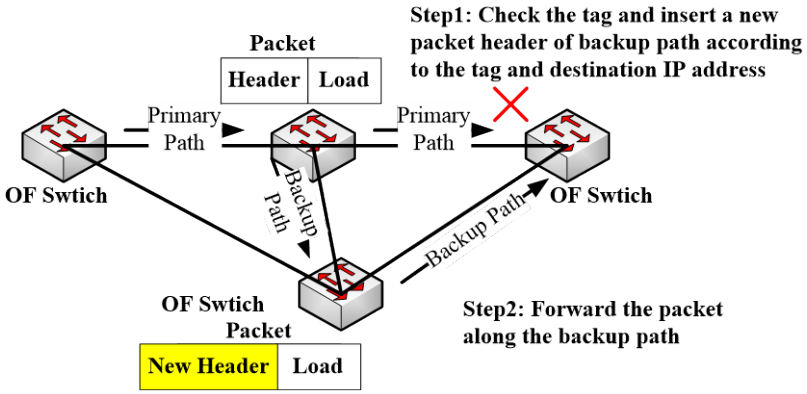


Fig. 7. The fast failure recovery process

3.4 Example

Consider the topology as **Fig. 8**, there are two working flows from *S* to *T*, which marked as flow1 and flow2. The transmit rate of flow1 is 50Mbps, and flow2 is 30Mbps. The working path is $S \rightarrow D \rightarrow T$. Every port of switch is identified by port id as **Fig. 8** shows, then we can demonstrate our fast failure recovery method as following steps:

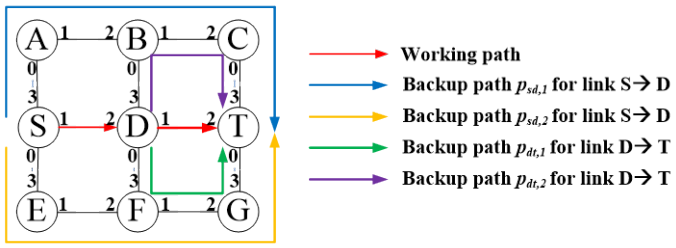


Fig. 8. Topology of network

Step1: Backup path planning. There are two links have to be protected: $S \rightarrow D$ and $D \rightarrow T$. We presume two backup paths for each link are used. Then the controller performs the backup paths planning algorithm for links $S \rightarrow D$ and $D \rightarrow T$ according to the network status. We assume that after backup paths planning process, two backup paths $S \rightarrow A \rightarrow B \rightarrow C \rightarrow T$ and $S \rightarrow E \rightarrow F \rightarrow G \rightarrow T$ are chosen for $S \rightarrow D$, and are tagged by $p_{sd,1}$ and $p_{sd,2}$ separately. Similarly, The paths $D \rightarrow B \rightarrow C \rightarrow T$ and $D \rightarrow F \rightarrow G \rightarrow T$ are chosen for $D \rightarrow T$ and are tagged by $p_{dt,1}$ and $p_{dt,2}$. Then the controller installs flow entries on switch S and D, which match packet's VUP field and update the packet header.

Step2: Backup path selection. The available bandwidths of backup paths $p_{sd,1}$ and $p_{sd,2}$ at switch S are assumed to be 70Mbps and 60Mbps. After the backup paths selection process, we assume that the controller assigns flow1 to path $p_{sd,1}$ and flow2 to path $p_{sd,2}$ at switch S according to the available bandwidth of backup paths and the demands of flows. Then the first hop information of flow1 is organized as "$1p_{sd,1}$". The "1" represents the output port ID and the "$p_{sd,1}$" represents the backup path id. In addition, if the backup path tag is X, then there is no backup path for the flow. After the same processes in switch D, we assume that the packet header of flow1 is "$1p_{sd,1}1p_{dt,1}$" and that of flow2 is "$1p_{sd,2}1p_{dt,2}$".

Step3: Fast link failure recovery. We assume link $D \rightarrow T$ is failed at sometime and then fast recovery process starts. We take flow1 as example. At switch S, it checks the outer most VLAN tag which is "$1p_{sd,1}$". After matching process, switch deletes the tag "$1p_{sd,1}$" and the packet header changes from "$1p_{sd,1}1p_{dt,1}$" to "$1p_{dt,1}$". Then switch S outputs the packet through port1 as there is no link failure at link $S \rightarrow D$. At switch D, the packet's header is "$1p_{dt,1}$" and port1 is failed. Then switch D checks the backup path ID which is $p_{dt,1}$. According to the flow entries installed in step1, switch D pops the VLAN tag "$1p_{dt,1}$" and pushes VLAN tags "$0X$", "$1X$", "$3X$" , which represent the backup path 1: $D \rightarrow B \rightarrow C \rightarrow T$ at switch D. Then the packet is checked again and output through port3. The same actions are also performed on flow2 so that the interrupted flows can bypass the failed link.

4. Backup Path Planning and Selection

4.1 Backup Path Planning

The network can be presented by a bidirectional graph $G(V,E)$, V is the set of nodes in graph G and E is the set of links in graph G. Let's define the following parameters:

D_{sd}	The set of paths from source s to destination d;
B	The set of backup paths which are selected;
F	The set of flows need to reroute;
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 to B, and 0 otherwise;
c_p	The available bandwidth of path p;
(k,o)	The failed link (k,o);
f_i	The flow i;
$c(f_i)$	The bandwidth demand of flow f_i;
$r_i^{j,h}$	Indication function which is equal to 1 if flow f_i is assigned to path j at hop h, and 0 otherwise;
$d_{p,h}$	The total bandwidth demand of flows assigned to path p at hop h;
p_v	The number of ports of switch v;
Num_p	The upper bound of the number of backup paths for every link;

In backup path planning process, we have to find a set of backup paths which can meet the following requirements: (1) Any one of the backup path should not contain the failure node and be loop free;(2) The backup paths should be link disjoint; (3)The number of backup paths for every link should be no more than the upper bound Num_p; (4)The backup paths should provide as much available bandwidth as possible; (5) The backup paths should be as short as possible. So we can formalize the problem to a multi objective optimization problem as (1) shows:

$$V = \min[\sum_{p\in D_{sd}}\sum_{ij\in E} e_{ij}^{p} t_p^B, -\sum_{p\in D_{sd}} t_p^B c_p], \forall (i,j)\in E, \forall p \in D_{sd} \quad (1)$$

In practice, available bandwidth that needs to reroute flows should have a lower bound. So we introduce parameter $\gamma(\gamma \geq 1)$, and define the lower bound as $\gamma\sum_i c(f_i)$. Then the problem is transformed to K-shortest disjoint paths with bandwidth guarantee (K-SDPBG). So we can have the multi constrained 0-1 integer linear programming model as follows:

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

$$s.t \quad \sum_p t_p^B c_p \geq \gamma \sum_i c(f_i), \forall p \in D_{sd} \quad (2)$$

$$\sum_p e_{ko}^p = 0, \forall p \in D_{sd} \quad (3)$$

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

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

$$\sum_p e_{ij}^p \leq 1, \forall i \in N, \forall p \in D_{sd}, \forall (i,j) \in E \quad (6)$$

$$\sum_p t_p^B \leq Num_p, \ \forall i \in N, \forall (i,j) \in E \quad (7)$$

The formula (2) guarantees that the available bandwidth provided by backup paths is more than the lower bound. The formula (3) guarantees that any of the backup paths doesn't contain the failure link. The loop free of backup paths is obtained by formula (4) and (5). And formula (6) represents that all the paths are link disjoint. The upper bound of backup paths number is defined in formula (7). And Num_p should be determined by in-degree of destination node, minimal out-degree of nodes in working path and the bit length of backup path tag, which can be presented as (8):

$$Num_p = \min\{\min\{p_i^{out}\}, p_d^{in}, 2^n\} \quad (8)$$

This problem is NP hard and we propose a heuristic algorithm consisted of two procedures to solve the problem. First, we modify the Dijkstra algorithm to find the *k* disjoint shortest paths from *s* to *d* as Procedure1 shows. In Procedure1, we find the shortest path *p* from *s* to *d* as normal Dijkstra algorithm in every loop, which is showed in line 1~21. Then we delete the links contained in path *p* to guarantee that all paths selected are link disjointed as line 22 shows. Until there is no path can be found from *s* to *d* in *G*, the Procedure1 stops.

```
Procedure 1 ModifiedDijkstra(graph, start, destin)
1:    do
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
18:    flag = startTo(destin)
19:    if flag!=INF  then
20:     pathSet.add(pathListMap.get(destin))
21:     end if
22:    UpdateGraph(graph, pathListMap.get(destin))
23:   end do
24:   while flag!=INF
25:   end while
```

Second, we have to select *n* backup paths from the path set formed in Procedure1. The paths selected should have minimal hops in total and can provide sufficient available bandwidth. To solve the problem, we design a dynamical programming algorithm as Procedure2 shows. Backtracking is used to solve this problem and the state transmission formula can be presented as *f[i][a][b] = min{f[i-1][a][b],f[i-1][a-a[i]][b-b[i]]}*, which is shown in line 10~16.

```
Procedure2 BackupPathSet (i, bandW, num)
1:    if i==0
2:     if bandwidth[0]>bandW && num>0
3:      return length[0]
4:     end if
5:     else return INF;
6:     end else
7:    end if
8:    else
9:     int temp=INF
10:    if number>0
11:       if(BackupPathSet(i-1,bandW-bandwidth[i],num-1) ==0)
12:      temp = BackupPathSet(i-1,bandW,num)
13:     end if
```

```
14:     else     temp=Minimum(BackupPathSet(i-1,bandW-bandwidth[i],num-1),
        BackupPathSet(i-1,bandW,num))
15:     end else
16:     end if
17:   return temp
18:  end else
```

In practice, it is not necessary to run backup path planning process frequently because that with the change of network status, the backup paths may not be the optimal but can still satisfy the demand of flows. Therefore, we set a threshold $\tau\sum_i c(f_i)$ $(\tau \geq 1)$ to trigger the backup path planning process. When $\sum_p t_p^B c_p < \tau\sum_i c(f_i)$ stands, the controller restarts the process to find more optimal backup paths.

4.2 Backup Path Selection

For the congestion avoidance during the flow rerouting, we have to distribute flows into different backup paths according to the demands of flows and the available bandwidth every path has. First, we define an assignment of path *k* at hop *h* as *P(k)* shown in (9):

$$P_h(k) = \{r_0^{k,h}, r_1^{k,h}, \cdots, r_n^{k,h}\} \tag{9}$$

And we define the cost function of *P(k)* as follows:

$$g_k(P_h(k)) = \frac{(h_{p_k})^{\alpha}}{c_{p_k} - \sum_0^n r_i^k c(f_i)} \tag{10}$$

The α is the adjustment parameter and h_{p_k} is the length of path p_k. When α is large, the flow tends to select a shorter path, otherwise, the flow tends to select a path with more available bandwidth. Then we can have the mathematical model as follows:

$$\max \min\{g_k(P(k))\}$$

$$s.t. \quad d_{p,h} = \sum_i r_i^{p,h} c(f_i)\ ,\ \forall f_i \in F, \forall p \in B \tag{11}$$

$$\sum_i c(f_i) = \sum_p d_{p,h}, \forall\ f_i \in F, \forall p \in B \tag{12}$$

$$\sum_i r_i^{p,h} = 1\ , \quad \forall p \in B \tag{13}$$

$$c_p > \sum_i r_i^{p,h} c(f_i)\ , \forall p \in B, \forall f_i \in F \tag{14}$$

Formula (11) and (12) guarantee the flow consistence. Formula (13) makes sure that a flow can be assigned into at most one path. Formula (14) represents that the bandwidth requirements of flows assigned to a path should not more than the bandwidth available. It is easy to prove that this problem has the property of optimal substructure. So we can use greedy algorithm to solve this problem as Procedure3 shows. In every loop, we find flow with maximal rate and path with minimal value of cost, and then assign the flow to the path. When there is no flow that is not assigned, algorithm stops.

```
Procedure3  FlowAlocate(FlowRate,policy,path)
1:      while set.size() != FlowRate.length()
2:        int max = findMaxFlowRate(FlowRate)
3:        int min = findMinCostFunction(path)
4:        List<Integer> assignment = policy.get(min)
5:        assignment.add(max)
6:        policy.put(min, assignment)
7:        set.add(max)
8:      end while
```

5. Implementation

We build our method on an online test bed supported by SDNLAB[27] to verify the effectiveness of our method. The experimental network shown in **Fig. 9** is consisted of a software controller installed in Ubuntu-14.04 and five software openflow-enabled switches. The controller used is OpendayLight Desktop Litmus and the switch used is Open vSwtich 2.3.0. We realize the source routing SDN and carry out experiments on failure recovery time.

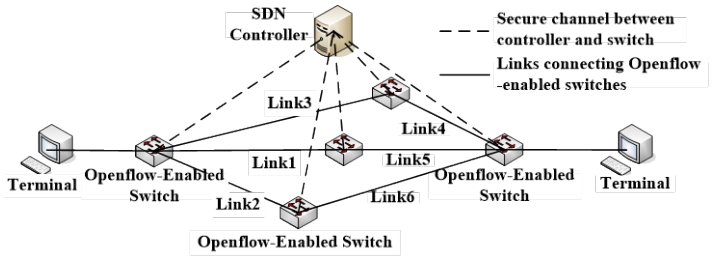


Fig. 9. Experimental network topology

For the sake of demonstration, we compare our method with segment protection which performs local failure recovery and has the minimal recovery time. In the segment protection, the recovery time consists of two parts as [28] demonstrates:

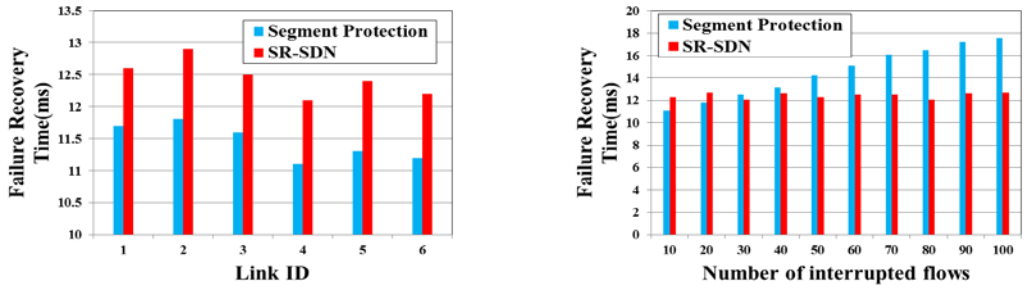
$$T_{recovery} = T_{detection} + \sum_{f} T_{handover,f} \quad (15)$$

The $T_{detection}$ is the failure detection time using per-link-BFD introduced in [11] which can realize failure detection time below 10ms in our schema. $T_{handover,f}$ is the time to activate the backup path of flow f in switch.

In the method we propose, the recovery time can be presented as (16):

$$T_{recovery} = T_{detection} + \sum_{f} T_{update,f} \quad (16)$$

The $T_{update,f}$ is the time to activate the backup path and modify the source routing header of packet. The $T_{update,f}$ may be larger than the $T_{handover,f}$ as the former has to perform additional actions to modify the packet header. However, in the open vSwtich we carry out evaluation, the difference is very small even the packet number is very large and the time of a switch to modify the alive-status of a group table is less than 1ms. Then we evaluate the failure recovery time under different conditions and the result is shown in **Fig. 10**.



(a) Failure recovery time on different link ID

(b) Failure recovery time on number of disrupted flows

Fig. 10. Failure recovery time result

From formula (15) and (16), it can be seen that the failure recovery time of segment protection and SR-SDN is independent of network scale. Furthermore, the **Fig. 10(a)** demonstrates that the failure recovery time of both methods has no relation with the location of the failure point. In **Fig. 10(b),** as the number of flows increases, the recovery time increases in segment protection method because there are more group table need to modify. However, the increasement is so slow that it still can achieve failure recovery within 50ms even dealing with large quantities of group tables. At the same time, the recovery time of SR-SDN is almost static and independent of the number of interrupted flows, as the number of group tables need to modify is static. So we can conclude that the method proposed in this paper can realize minimal and nearly static failure recovery time as segment protection and other methods which perform local link failure recovery.

Additionally, the sampling interval and the delay between switch and controller may influence the performance of our backup path planning and selection algorithms which make decision depending on the real time network status. To solve this problem, we average the network status information gathered from switches as formula (17) shows:

$$Avg_n = wAvg_{n-1} + (1-w)C_n \quad (17)$$

The Avg_{n-1} is the average value after *(n-1)*th computation. C_n is the *n*th sampling value of network status and *w* is the weigth value. With the processing of formula (17), the controller makes decision according to the trend of available bandwidth of the backup paths rather than the accurate value so that the mistakes caused by delay can be reduced. However, how to get the accurate value of *w* is remained to be researched and *w*=0.6 is recommended through several experiments.

6. Evaluation

In this section we derive analysis and simulations to evaluate the overhead and performance of the method this paper proposes.

6.1 Overhead Analysis

(1) Flow Entries Needed to Build Backup Paths

TCAM, which is used to store flow entries, is the most valuable resource of the switch. In traditional link protection methods, the TCAM may be exhausted rapidly because they need

too many entries to build backup paths for large quantities of flows.

Segment protection is a typical link protection method which has been widely used and ITP method introduced in [3] can significantly reduce the flow entry consumption. So in this section, we evaluate the flow entry consumption of our method and compare our method with segment protection and the ITP method. Consider a n*n matrix network like **Fig. 11** shows. The source node and the destination node can form a i*j submatrix topology and we presume that the working path is always the border of the submatrix as dashed line shown in **Fig. 11**.

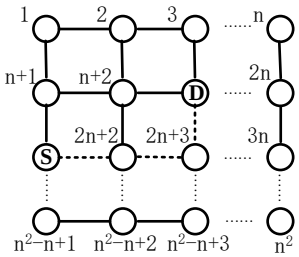


Fig. 11. n*n matrix network

It has been demonstrated in [1] and [3] that the total flow entries of segment protection can be presented as **(18)**:

$$F_{segment\_protection} = \sum_{i=2}^{n}\sum_{j=1}^{i}(N_{i,j} \times M_{n,i,j}) \tag{18}$$

And $N_{i,j}$, $M_{n,i,j}$ can be expressed by (19) and (20) according to [3]:

$$N_{i,j} = \begin{cases} 2j + 4i - 4; j < 3 \\ 4i + 4j - 8; j \geq 3 \end{cases} \tag{19}$$

$$M_{n,i,j} = (n-i+1) \times (n-j+1) \times \sigma_{i,j}; \sigma_{i,j} = \begin{cases} 8, if\ \ j \neq 1, i \neq j \\ 4, if\ \ i = 1\ or\ i = j \end{cases} \tag{20}$$

In independent transient plane (ITP) method, the total number of flow entries can be presented as (21)~(23):

$$F_{ITP} = F_{working\_path} + F_{transient\_plane} \tag{21}$$

$$F_{transient\_plane} = n^2 \times [2n^2 + n - 2] \tag{22}$$

$$F_{working\_path} = \sum_{i=2}^{n}\sum_{j=1}^{i}[(i + j - 1) \times M_{n,i,j}] \tag{23}$$

In our method, the total flow entries can be expressed as (24):

$$F_{SR-SDN} = F_{working\_path} + F_{transient\_plane} \tag{24}$$

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 (25):

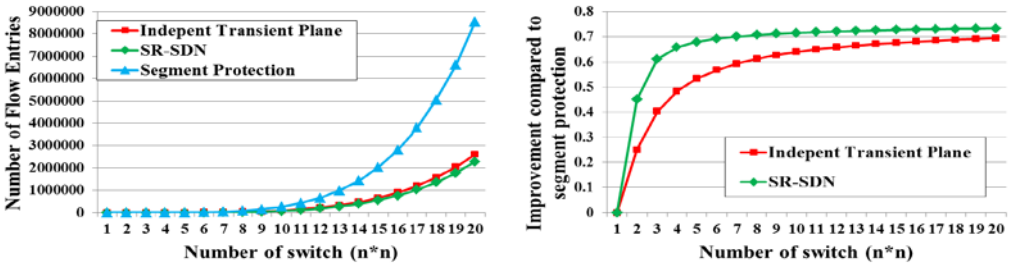
$$F_{working\_path} = n^2 \times 2p \tag{25}$$

p is the number of ports in every switch and $2p$ is the sum of matching flow entries and fast failover flow tables. 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 (26):

$$F_{backup\_path} = \sum_{i=2}^{n}\sum_{j=1}^{i}[(i+j-1)\times M_{n,i,j}] \quad (26)$$

The total flow entries of three methods are shown in **Fig. 12(a)**.

It can be seen that our method has the minimal flow entry consumption among three methods and the advantage is expanding with the increasment of network scale. And the improvement of ITP and SR-SDN compared with segment protection is shown in **Fig. 12(b)**. It can be demonstrated that our method can realize less flow entry overhead compared to ITP.



(a) Total number of flow entries (b) Improvement compare to segment protection

Fig. 12. The analysis of flow entries

(2) The Complexity Of Algorithms

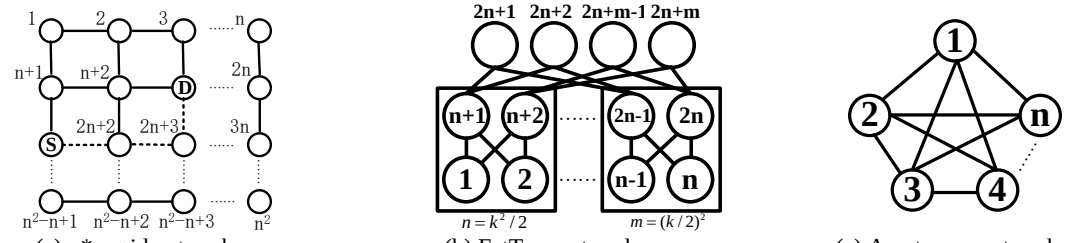
The complexity of Dijkstra algorithm in Procedure1 is $O(n^2)$ and can be optimized to $O(ElogV)$. E is the number of links in graph G and V is the number of vertexes. K is the number of backup paths at every hop and N is the length of working path. In the worst case, the complexity of Procedure1 is $O((E^2+E)/2*logV)$. And the complexity of procedure2 is $O(E!/(K!(E-K)!))$. So the complexity of backup paths planning algorithm is presented as (27):

$$O(Procedure1 + Procedure2) = O(\frac{(E^2+E)N}{2}\log V + \frac{NE!}{K!(E-K)!}) \quad (27)$$

The complexity of Procedure3 can be presented as (28), and M is the number of flows to reroute:

$$O(Procedure3) = O(\frac{NM(M+2K+1)}{2}) \quad (28)$$

Several symmetrical and unsymmetrical topologies are used as test platforms to evaluate the time consumption of backup path planning algorithms in real network, such as grid network, FatTree, any-to-any network, NSFNet and etc as Fig.13 shows. Because the execution of the backup path planning algorithm is independt at every hop, so we test the algorithm only at one node for the sake of simplicity. We realize algorithms with Java and carry out experiments at PC which runs Ubuntu-14.04 with a CPU of Core i5-4210m 2.6Hz and 4GB RAM. The links in every topology are bidirectional and the available bandwidth is randomly distributed in (0,100). There are 10 flows and the rates of which are randomly distributed in (0,10). The experiment is repeated 50 times and the result is averaged.



(a) n*n grid network (b) FatTree network (c) Any to any network

(d) m*n leaf-spine network (e) Polska (f) NSFNet

Fig. 13. Test topologies

To evaluate the efficiency of algorithm, we define the optimal ratio as follows:

$$Optimal\ Ratio = \frac{\sum_{p}\sum_{ij} e_{ij}^{p} t_{p}^{B}\ of\ optimal\ solution}{\sum_{p}\sum_{ij} e_{ij}^{p} t_{p}^{B}\ of\ our\ algorithm} \tag{28}$$

Then we evaluate the optimal ratio and the time consumptions of our algorithms. The test result is demonstrated in **Table 1**.

Table 1. The performance of algorithms with K=2

Topology	Optimal Ratio (%)	Time Consumption(ms)
10*10 Grid network	89.22%	187ms
FatTree with n=8,m=4	86.21%	15ms
Any-to-Any network with 100 nodes	92.35%	188ms
m*n Leaf-Spine network with m=10,n=20	95.22%	156ms
Polska	91.83%	5ms
NSFNet	98.84%	5ms

Through the result in **Table 1**, we can conclude that: our algorithm can get nearly optimal results in an acceptable time so that it is efficient and can be applied to most of the networks.

6.2 Performance Analysis

(1) Simulation Setup

To evaluate the performance of the method, we carry out a simulation with OMNET++ 4.5 and INET. The simulation topology is a 5*5 grid network as **Fig. 14** shows. The working flow is from cluster1 to cluster2 and the flow can be classified into 9 sub flows depending on different source and destination. In order to generate background flows, every host except hosts of two clusters randomly send packet to another host with a shortest path.The SDN architecture follows the instruction of [29] and uses out-of-band control mode for simplicity. Every switch port uses FCFS and has a maximum transmit rate of 1Mbps. The working path is the shortest path from *S* to *T* and the failed link is showed in **Fig. 14**.

In this section, besides segment protection, multipath rerouting (MPR) method introduced in [2] is chosen as the comparison method because it is a typical method to solve the congestion problem and has good load balancing performance in link failure recovery. In segment protection, the backup path is the shortest path from failure node to the destination. In multipath rerouting and source routing SDN, we choose 2 backup paths for every link. And the parameters we use are shown in **Table 2**.

Table 2. Parameters used in simulation

Parameter	Value	Description
γ	1.2	Parameter to determine the lower bound of available bandwidth provided by backup paths in formula (2).
τ	1	Parameter to determine the threshold of available bandwidth provided by backup paths to re-activate backup path planning.
ω	0.6	The weight to average the network status information in formula (17).
t	0.01s	The sampling interval of the network status information during simulation.

The simulation lasts 100s and link failure occurs at 20s. In practice, the fast link failure recovery time may be very short because the controller may repair the failed link or build a new end-to-end path very soon after detecting the link failure. For the sack of demonstration, we presume that the controller doesn't react to the link failure so that the fast link recovery process can last long enough to evaluate its performance.

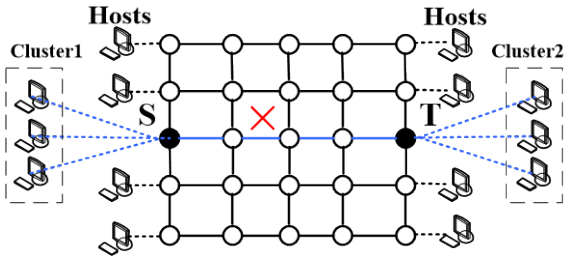
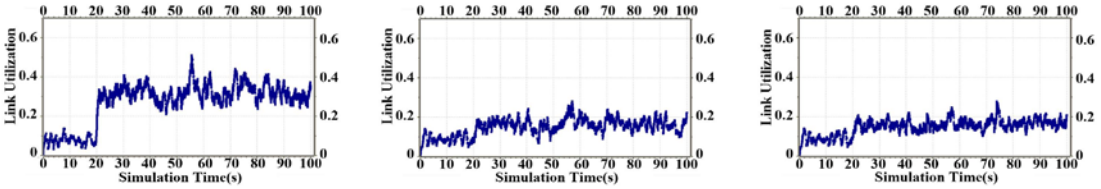


Fig. 14. The simulation topology

(2) Result Analysis

In order to verify the effectiveness of the proposed algorithm, two different traffic patterns are simulated: light load and heavy load. Under the light load situation, data packet arrival interval obeys the exponential distribution with λ=0.1, and the average link utilization of the network is about 10%~20%. Under the heavy load situation, the interval of data packet arrival time obeys the exponential distribution with λ=0.01. The average link utilization of working path is about 60%~80%.

Under the light load situation, the average link utilization curves of the backup paths in the three methods are shown in **Fig. 15**:



(a) Segment Protection (b) Multipath Rerouting (c) Source Routing SDN

Fig. 15. Link Utilization in Light Weight Situation

According to the data recorded in simulation, the average link utilization of segment protection is 38.76% after rerouting process, which is higher than the 18.32% of MPR and 16.54% of SR-SDN. And the CDF of link utilization of three methods are shown in **Fig. 16**. Compared with MPR, the link utilization of SR-SDN is about the same because the difference of the backup paths' load is very small after the processing of formula (17), which makes the controller rarely restarts the backup path planning or reassign the flows. Thus, it can make algorithm avoiding unnecessary adjustment when available bandwidth is sufficient. During the simulation, the backup path planning is never re-activated and the number of flow redistribution is 4. In summary, only with a few adjustments, SR-SDN can achieve better load balancing performance with an improvement of 57.33% comparing with segment protection and 9.7% comparing with MPR under the light load situation.

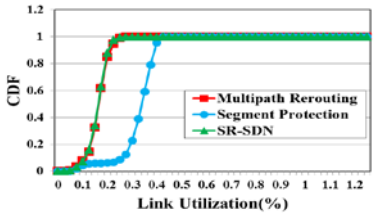
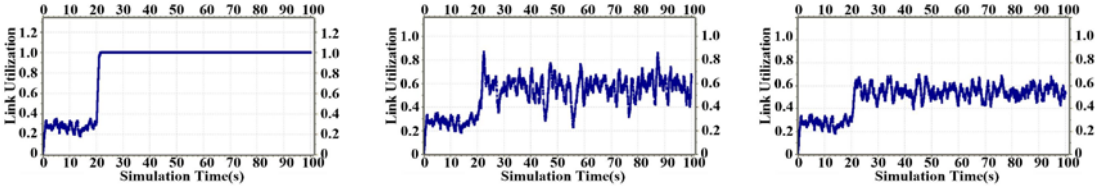


Fig. 16. The CDF of link utilization in light load situation

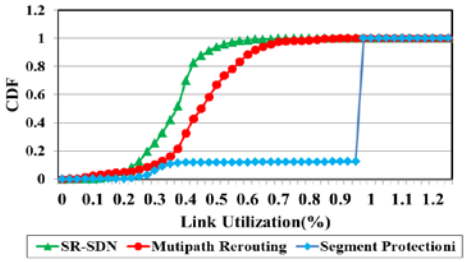
Under the heavy load situation, the link utilization curves of three methods are shown in **Fig. 17**.



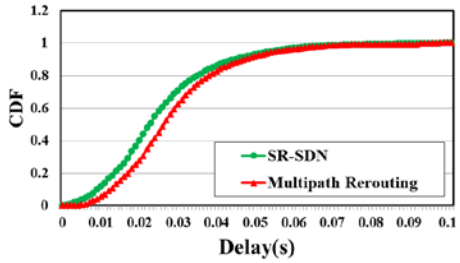
(a) Segment Protection (b) Multipath Rerouting (c) Source Routing SDN

Fig. 17. The link utilization during flow rerouting process

When traffic load is heavy, segment protection is totally not working as **Fig. 17(a)** shows because backup path can't provide sufficient bandwidth to reroute interrupted flows. On the contrary, MPR and SR-SDN can balance the load of different backup paths to make full use of available bandwidth provided by network. And the CDF of link utilization of three methods are shown in **Fig. 18(a)**. It can be seen that SR-SDN can significantly reduce the average link utilization compared with segment protection and MPR. We call it congestion when link utilization is over 80%. And the SR-SDN reduces the congestion possibility from 87.1% of segment protection and 1.06% of Multipath Rerouting to zero.



(a) The CDF of link utilization in heavy load situation



(b) The CDF of end-to-end delay suffered by rerouted packets

Fig. 18. Cumulative distribution function of link utilization and delay

With the improvement of the link utilization, rerouted packets in SR-SDN can achieve better delay performance and reduce the average delay about 21.2% compared with MPR as **Fig. 18(b)** shows. In summary, the SR-SDN method proposed in this paper can achieve fast failure recovery with much less overhead and significantly reduce the possibility of congestion during flow rerouting process.

7. Conclusion

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, with source routing method, we store the working path and backup path information in packet header rather than flow entries installed in switches. So it can achieve fast link failure recovery with a very low flow entry consumption; Third, through source routing SDN, we decouple the flow from the route that they transmit, so that the working paths and backup paths for flows can be easily reconstructed according to the network status and more flexible traffic engineering for link protection can be realized. Finally, we design congestion aware backup path planning and selection algorithms taking into account the variability of network status so that the performance of the rerouting algorithms can be further improved. 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.

However, there are still some issues to be solved in implementation. For example, the source routing SDN proposed in this paper may introduce overhead to the packet header and have some scalable problems when applied in large scale network. Moreover, the sampling interval and the acquisition delay of network status may cause errors of the backup paths planning and selection algorithms.

Our next step research will focus on solving the problems mentioned above and mainly includes following aspects: First, we will develop customized source routing SDN mechanism to minimize the overhead of packet header and eliminate the scalability issues; Second, we will further optimize the proposed mechanism and algorithms to minimize the impact of sampling interval and acquisition delay of network status; Third, we will further explore the application scenarios of proposed method to enhance its practicality.

References

[1] 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*, vol. 5, no. 9, pp. 1066-1075, September, 2013. Article (CrossRef Link)

[2] A. Capone, C. Cascone, A. Q. T. Nguyen and B. Sansò, "Detour planning for fast and reliable failure recovery in SDN with OpenState," in *Proc. of 11th Int. Conference on Design of Reliable Communication Networks* , pp. 25-32, March 25-27, 2015,.Article (CrossRef Link)

[3] N. Kitsuwan, D. B. Payne and M. Ruffini, "A novel protection design for OpenFlow-based networks," in *Proc. of 16th Int. Conference on Transparent Optical Networks*, pp. 1-5, July 6-10, 2014. Article (CrossRef Link)

[4] R. M. Ramos, M. Martinello and C. Esteve Rothenberg, "SlickFlow: Resilient source routing in Data Center Networks unlocked by OpenFlow," in *Proc. of 38th IEEE Conference on Local Computer Networks*, pp. 606-613, October 21-24, 2013. Article (CrossRef Link)

[5] S. Cevher, M. Ulutas, S. Altun and I. Hokelek, "Multi Topology Routing based IP Fast Re-Route for Software Defined Networks," in *Proc. of 2016 IEEE Symposium on Computers and Communication (ISCC)*, pp. 1224-1226, June 27-30, 2016. Article (CrossRef Link)

[6] B.Jenkins, D.Brungard, M.Betts, N.Sprecher, S.Ueno, "MPLS-TP Requirements", RFC5654, IETF, 2009. Article (CrossRef Link)

[7] W. Braun and M. Menth, "Scalable resilience for Software-Defined Networking using Loop-Free Alternates with loop detection," in *Proc. of 1st IEEE Conference on Network Softwarization*, pp. 1-6, April 13-17, 2015. Article (CrossRef Link)

[8] Adami, D., et al, "Class-based traffic recovery with load balancing in software-defined networks," *IEEE GLOBECOM 6th Management of Emerging Networks and Services Workshops*, pp.161-165, December 8-12, 2014. Article (CrossRef Link)

[9] D.Katz, D.Ward, "Bidirectional Forwarding Detection," RFC-5880, IETF, 2010. Article (CrossRef Link)

[10] Sharma, Sachin, et al, "OpenFlow: Meeting carrier-grade recovery requirements ," *Computer Communications*, vol. 36, no. 6, pp.656-665, 2013. Article (CrossRef Link)

[11] 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*, pp. 61-66, September 1-4, 2014. Article (CrossRef Link)

[12] S. S. W. Lee, K. Y. Li, K. Y. Chan, G. H. Lai and Y. C. Chung, "Software-based fast failure recovery for resilient OpenFlow networks," in *Proc. of 7th Int. Workshop on Reliable Networks Design and Modeling*, pp. 194-200, October 5-7, 2015. Article (CrossRef Link)

[13] 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," in *Proc. of 10th Int. Conference on Design of Reliable Communication Networks*, pp. 1-8, March 31-April 3, 2014. Article (CrossRef Link)

[14] Kotani, Daisuke, and Y. Okabe, "Fast Failure Detection of OpenFlow Channels," in *Proc. of The Asian Internet Engineering Conference 2015*, pp. 32-39, November 18-20, 2015. Article (CrossRef Link)

[15] S. Sharma, D. Staessens, D. Colle, M. Pickavet and P. Demeester, "Enabling fast failure recovery in OpenFlow networks," in *Proc. of 8th Int. Workshop on Design of Reliable Communication Networks*, pp. 164-171, October 10-12, 2011. Article (CrossRef Link)

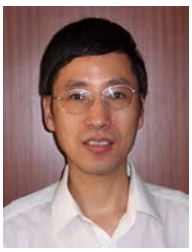
[16] R. Kanagavelu, L. N. Mingjie, K. M. Mi, B. S. Lee, Heryandi and H., "OpenFlow based control for re-routing with differentiated flows in Data Center Networks," in *Proc. of 18th Int. Conference on Networks*, pp. 228-233, December 12-14, 2012. Article (CrossRef Link)

[17] S. Astaneh; S. Shah Heydari, "Optimization of SDN Flow Operations in Multi-Failure Restoration Scenarios," *IEEE Transactions on Network and Service Management* , vol.6, no.99, pp.1-11, June, 2016. Article (CrossRef Link)

[18] S. Sharma, D. Staessens, D. Colle, M. Pickavet and P. Demeester, "Fast failure recovery for in-band OpenFlow networks," in *Proc. of 9th Int. Conference on Design of Reliable*

Communication Networks, pp. 52-59, March 4-7, 2013. Article (CrossRef Link)
[19] Sahri, N. M., and K. Okamura, "Fast failover mechanism for software defined networking: OpenFlow based," *International Conference on Future Internet Technologies ACM*, pp. 1-2, June 18-20, 2014. Article (CrossRef Link)
[20] "OpenFlow Switch Specification Version 1.5.0", Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.pdf , Open Network Foundation, December, 2014.
[21] A.Atlasand, A.Zinin, "Basic Specification for IP Fast Reroute: Loop-Free Alternates, " RFC5286, IETF, 2008. Article (CrossRef Link)
[22] C. Y. Chu, K. Xi, M. Luo and H. J. Chao, "Congestion-aware single link failure recovery in hybrid SDN networks," in *Proc. of 2015 IEEE Conference on Computer Communications*, pp. 1086-1094, April 26 – May 1, 2015. Article (CrossRef Link)
[23] C. Y. Chu, K. Xi, M. Luo and H. J. Chao, "Congestion-aware single link failure recovery in hybrid SDN networks," in *Proc. of 2015 IEEE Conference on Computer Communications (INFOCOM)*, pp. 1086-1094, April 26-May1, 2015. Article (CrossRef Link)
[24] M. Soliman, B. Nandy, I. Lambadaris and P. Ashwood-Smith, "Exploring source routed forwarding in SDN-based WANs," in *Proc. of 2014 IEEE International Conference on Communications* , pp. 3070-3075, June 10-14, 2014. Article (CrossRef Link).
[25] Soliman, M., Nandy, B., Lambadaris, I., & Ashwood-Smith, P. "Source routed forwarding with software defined control, considerations and implications," in *Proc. of 2012 ACM Conference on CONEXT Student Workshop*, pp.43-44, December 4-9, 2012. Article (CrossRef Link)
[26] Jyothi, Sangeetha Abdu, M. Dong, and P. B. Godfrey. "Towards a flexible data center fabric with source routing," in *Proc. of 2015 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, pp.1-8, June 17-18 , 2015. Article (CrossRef Link).
[27] "Online SDN Experimental Plateform," Available: http://www.sdnlab.com/experimental-platform/, May, 2016.
[28] Pankaj Thorat, et al. "Optimized self-healing framework for software defined networks," in *Proc. of 9th Int. Ubiquitous Information Management and Communication*, pp. 1-6, January 8-10, 2015. Article (CrossRef Link)
[29] Klein, Dominik, and M. Jarschel, "An OpenFlow extension for the OMNeT++ INET framework," in *Proc. of 6th Int. ICST Conference on Simulation TOOLS and Techniques*, pp. 322-329 , March 6-8, 2013. Article (CrossRef Link).

Liaoruo Huang received his B.E. degree in information security from Xidian University, Xian, China, in 2011. And he received the M.S. degree in information security science and technology from PLA University of Science and Technology, Nanjing, China, in 2014. He is currently working toward Ph.D. degree in College of Communications Engineering, PLA University of Science and Technology. His research interests include software-defined network, next generation networks and wide area network.



Qingguo Shen received his Ph.D. degree from PLA University of Science and Technology, Nanjing, China, in 1994. He is currently a full professor in College of Communications Engineering,, PLA University of Science and Technology. His research interests include next generation networks, M2M and wireless networks.

Wenjuan Shao received her B.E. degree in computer communication and M.S. degree in computer networks from Beijing University of Posts and Telecommunications, Beijing, China, in 2001 and 2004 respectively. She worked as an engineer for the China Mobile Communications Corporation from 2004 to 2012. In 2007, she became a senior lecture at Zijin College, Nanjing University of Science and Technology. She is currently working toward Ph.D. degree in College of Communications Engineering, PLA University of Science and Technology. Her research interests include D2D, social aware network and software defined network.