# Backup-Resource Based Failure Recovery Approach in SDN Data Plane

Shujuan Zhang, Ying Wang, Qichao He, Jinke Yu, Shaoyong Guo
State Key Laboratory of Networking and Switching Technology
Beijing University of Posts and Telecommunications
Beijing, China
Email: {zhangsj, wangy}@bupt.edu.cn

***Abstract*—Software Defined Networking (SDN) enables the underlying infrastructure to be abstracted from the network services and controlled by one or more controllers. If a link or a node fails, the switches that can detect the failure have to either inform controller to update flow tables or transform the data to pre-configured paths to recover the failure. However, existing failure recovery approaches mainly consider the recovery delay and packet loss, and ignore the storage resources consumption for backup paths in case of link or node failure. Moreover, the Ternary Content Addressable Memory (TCAM) that stores flow entries is expensive and limited with high-energy consumption. Thus in order to minimize the consumption of backup resources and meet the required failure recovery delay, a backup-resource based failure recovery approach is proposed. Two metrics are proposed to grade physical links, and three kinds of strategies for different graded links are provided, based on which the approach tries to use less flow entries to recover link failure and meets the required failure recovery delay, while guaranteeing the reliability of the network. Simulations show that backup-resource based approach can use as less flow entries as possible to ensure the performance of failure recovery and satisfy the required delay of important traffic at the same time. Moreover, the approach has good and steady performance in networks of different scales and connectivity.**

***Keywords—SDN; flow entry; failure recovery; data plane***

## I. INTRODUCTION

The research about Software Defined Networking (SDN) has got more and more attention from academia and industry recent years. One of the characteristics of the SDN is that it separates data and control functions of networking devices with a well-defined Application Programming Interface (API), which claims that the control of the network is realized by the centralized controller platform. In the architecture of the SDN, the existence of the control plane can make the network deployment and configuration more intelligent and simplified. With all complex functions subsumed by the controller, switches is mainly responsible for managing flow tables whose entries can only be populated by the controller [1].

Obviously, data plane is the key part for ensuring normal operation of the SDN network, thus one of the problems in SDN data plane is the failure recovery. Some approaches [2-3] propose reactive strategies to solve it. However, none of them take into account the latencies due to the communication of switches with the remote controller, which may affect the quality of service for the businesses with higher request of real-time and cause the loss of data packets [4-5].

With that in mind, there are a number of studies focusing on the fast failure recovery in SDN data plane. The approach of [6] use the Failover Group Tables proposed by the Openflow specification from the version of 1.3 to decrease the failure recovery time. In contrast, some approaches [7-8] extend Openflow protocol to eliminate the communication with remote controller when a link fails. Although it can ensure fast link failure recovery, the extension of Openflow protocol has an obvious drawback of complexity of implementation. The others [9-10] try to overcome the problem of latencies and complexity by setting priority for traffic or backup path. However, those proactive strategies against link failures have not been explicitly studied. Due to the increase of businesses volume and/or the scale of network, the calculation amount and the resources consumption of switches for failure recovery will increase. Furthermore, in Openflow-enabled network, flow entries are used to route the flows through its pre-defined paths. However, the flow entries are stored in Ternary Content Addressable Memory (TCAM), which is an expensive and limited hardware with high-energy consumption [11]. For example, N.Katta reported that TCAMs are 400 times more expensive [12] and 100 times more power consuming per Mbit than RAM-based storage [13]. Moreover, with the communication traffic increasing, the number of flow entries in switches will also augment. Eventually, the OpenFlow switch will run out of storage space and begin deleting the entries in TCAMs. Network latency and packet loss will deteriorate at the same time, and the network quality of services (QoS) will also drop [14].

In this paper, we propose a failure recovery approach based on backup resources to minimize the consumption of storage resources of switches, while meeting the required recovery delay when link fails. The contribution of this paper is threefold. First, we propose two metrics to determine the importance level of a link. Second, three kinds of strategies for different graded links are provided, based on which we give the formulation of the problem and design algorithms to solve the problem. Third, we conduct simulations to show the validity of our approach and assess the performance in networks of different scales and connectivity.

The rest of this paper is organized as follows. Section II discusses the related works. Section III analyzes and defines the problem. Section IV introduces the proposed approach.

Simulation results are presented in Section V and this paper is concluded in Section VI.

## II. RELATED WORK

At present, research about failure recovery strategy in SDN data plane can be classified into two categories: reactive and proactive, which can also be used into control traffic.

The study of [2] mainly considers reactive strategy, in which the controller is entitled to monitor link status in the network, and, in case of failure, it calculates a new path for the affected demand and replaces or removes flow entries in switches, accordingly. In [3] the authors presented reactive and proactive mechanisms for control and data traffic in the case of link failures of SDN. The results indicate that the failover time for proactive and reactive are about 45ms and 70-130ms, respectively. And this paper did not consider situations where the controller or switches themselves crash.

However, reactive recovery hardly achieves carrier-grade requirement in large scale networks [3], thus proactive recovery is studied pervasively for fast failover in SDN data plane. In [15], V. Padma and P. Yogesh propose link protection scheme by adding fast recovery mechanisms in the switch and controller. It avoids controller intervention in the case of single link network failures, which reduces not only recovery time but also the overhead of the controller. The simulation results showed that the scheme performs reasonably better than the existing scheme in terms of switchover time. However it increases the number of flow entries for backup path relatively. The authors of [7] proposed a proactive scheme with OpenFlow extension, called OpenState, which can autonomously adapt forwarding rules in a stateful fashion. The scheme discussed the computation of backup paths, including link congestion level, distance of the reroute point from the failure detection point, and level of sharing of backup paths by different flows. And the results were compared in three aspects with three different network topologies. The idea of extending OpenFlow protocol is also used in [8], the authors proposed an end-to-end path protection scheme by implementing a monitoring function in OpenFlow switches, which can autonomously react to failures by switching to a precomputed end-to-end backup path, thus the scheme not only can reduce processing load on the controller, but also achieve data plane fault recovery in a scalable way within 50ms. By using the OpenFlow's Fast Failover Group Table, the authors of [6] introduce a failover scheme with preconfigured primary and secondary paths. And the backup path is calculated from every intermediate switch to destination in case of link failure. If a switch has no feasible backup path, it will return packets to the previous switch by *crankback* routing. After restoring link functionality, the Group Table reverts to the primary path. In [9], the authors proposed a control application of SDN for class-based traffic recovery with load balancing, and recovery mechanisms have been implemented for different traffic class, ranging from reactive in case of Bronze traffic to 1+1 proactive for Gold traffic. Similar to [9], the authors of [10] also use the class strategy to grade the backup paths, in which segment protection is used in an Ethernet OpenFlow network for the case of link failure and interface failures. The working and backup paths are maintained at different priorities, and OpenFlow is extended to enable switches to locally react to connected failed links automatically without participation of the controller.

However, all these researches mainly consider the delay and packet loss during failure recovery in SDN. Different from these related works, we focus on minimizing the consumption of storage resources of switches for link failure recovering.

The proposed failure recovery approach in this paper can not only save storage resources, but also meet the required recovery delay. Moreover, it can behave well especially when the network scale is large.

## III. PROBLEM FORMULATION

Firstly, we give an instance of link failure recovery in SDN network. As shown in Fig.1, the network has eight OpenFlow switches and fourteen links, where the red line is the working path between the host *H1* and *H2*. There is no doubt that all flows going through the link <*S2,S5*> will be influenced in case of the failure on link <*S2,S5*>, and the switch S2 can detect the failure. There are four alternative backup paths in the network for the link failure recovery, which are distinguished by painting four different colors. And all of the backup paths are realized by configuring flow entries in related switches. The green one goes through four switches: S1, S3, S6 and S8, thus they all have one additional flow entry respectively. The purple one recovered by the detectable-switch S2, and the switches of S2, S4, S7 all have to store one flow entry respectively. Both the yellow and blue path are the backup paths from the detectable-switch S2 to switch S5 that is the next switch of S2 on working path, but the blue path has one more switch than the yellow one, thus the blue path will generate one more flow entry than the yellow path.

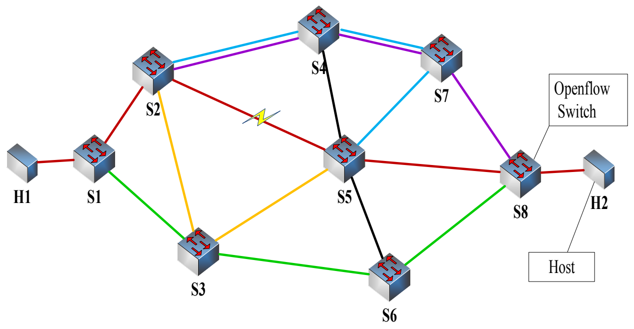

Fig.1. An example of SDN network with link failure

On the other hand, all these flow entries need to be pre-stored in related switches, and it may be a burden for the storage of flow tables. Since the capacity of TCAMs is limited, and the difficulty and complexity of searching specific flow entries will arise with the increase of the scale of the network. Thus the amount of switches belonging to backup path has a linear relationship with the resource consumption for link failure recovery, and our work focus on using as less flow entries as possible to achieve link failure recovery and aiming to meet required recovery delay.

Based on the above discussion, we formulate an SDN-network as an undirected weighted graph $G=(N,L)$, where $N$ represents the set of all the switches (e.g., OpenFlow switches), and $L$ represents the set of all the links among the switches. Let $F$ denotes the set of source-destination communication paths.

For a link *l*∈*L*, the total number of the flows going through the link *l* is denoted by *FN(l)*, the bandwidth utilization ratio is *BUR(l)*, and *BPN(l)* presents the number of backup paths going through link *l*. In fact, not all links in a network need to pre-configure the flow entries of backup paths, thus with the aim to minimize the resource consumption for failure recovery and satisfy the required delay at the same time, our first job is to find which links are the most important, and which links have merely influence in case of link failure. In general, we firstly need to confirm the importance level of links according to *FN* and *BUR*, then classify the set of *L* to three sets that have different priority, finally compute backup paths for each set according to the corresponding strategy we proposed.

## IV. THE PROPOSED BACKUP-RESOURCE BASED FAILURE RECOVERY APPROACH

In this section, we first explain how to grade the importance level of link. And based on the graded links, we define three different backup strategies for the three class links. Then we propose algorithms to find backup paths for links with different grades.

### *A. The Importance Level of Link*

In the real networks, the occurrence of link failure is random, and because of diverse links, the influence on the network or the communication between hosts is also different in case of link failure. In order to minimize the flow entries for pre-configuring backup paths and achieve required recovery delay in case of link failure, we propose metrics to classify the links in a network firstly.

Fig.2 represents the communication path of ten hosts in a SDN network, where five flows are differed by dotted lines with five colors. There are fourteen physical links between switches, which are represented by black solid line. And failure that happened on different links may cause diverse grade consequences for the network. For example, there are four flows get through link <*S2, S5*>, two flows through link <*S5, S8*> and <*S1, S2*>, one flow through link <*S3, S5*>, <*S4, S5*>, <*S5, S6*>, <*S5, S7*>, and the rest of links have no flow across at present. If link <*S2, S5*> fails, four flows will be influenced. Similarly, if other links fail, the flows going through the link will also be influenced. Furthermore, *S5* is a kernel node, which the five flows all run through it. Thus link <*S2, S5*> and node *S5* both are important for the present network.

Based on this, we rank the physical link so as to adopt diverse recovery strategies to implement failover with less resources. The evaluation criteria are formulated as follows:

$$FN^U = 1 - e^{(-FN)} \quad (1)$$

$$IMPL = \lceil 2(\alpha FN^U + (1-\alpha)BUR) \rceil \quad (2)$$

We select two metrics to present the importance level of links: *FN* and *BUR*, and the value of *BUR* has represented over the link, which is shown in Fig.2. Because of the relationship between *FN* and the importance level of link represented by *IMPL* are not merely the linear, and the effect to the link ranking will tend to be stable with the *FN* increasing, thus we use function (1) to transform the original *FN* to $FN^U$ to make it suitable for our research scenario. And function (2) represents the importance level of link, in which the coefficient $\alpha \in [0, 1]$ is used to adjust the weight of two metrics. Since $FN^U \in [0,1]$ and *BUR*∈ [0,1], thus the scope of *IMPL* is [0,2], and we set [0,0.6] as level 0, [0.7,1.3] as level 1, [1.4,2] as level 2 to normalize it. Furthermore, the higher the number is, the greater the level will be. Then the three kinds of grade are regarded as different importance level of links. From the explanation above, we know the value of *FN* and *BUR* of link <*S2, S5*> is 4 and 0.8 respectively, thus we can get the $IMPL_{(2,5)}$ of link <*S2, S5*>, which is 2. Similarly, we can calculate the real value of the importance level of link <*S4, S5*>, <*S3, S5*>, <*S5, S8*>, <*S2, S3*> and <*S1, S2*>, and they are 0.94, 1.14, 1.46, 0.02 and 1.26 respectively. After the normalization, the value of *IMPL* of each link is: 1, 1, 2, 0 and 1 respectively.

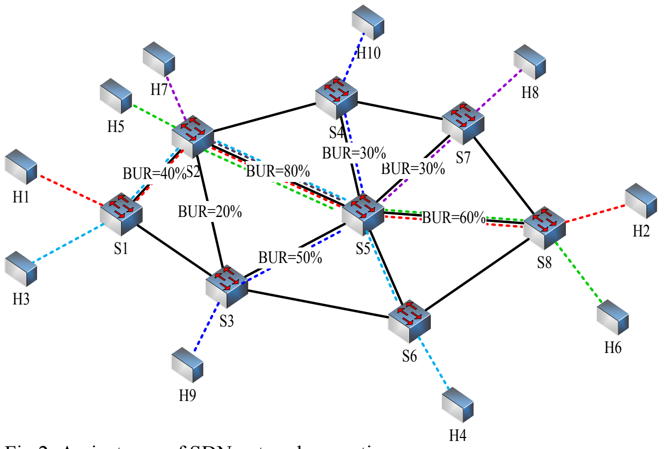

Fig.2. An instance of SDN network operation

### *B. Backup Strategy*

With the *IMPL* of each link, we make strategies to reduce the storage of flow entries in switches and satisfy the required delay. Before describing our backup-resource based approach, we propose three different backup strategies for the three kind of importance level of link:

*1) Double-Path Strategy:* The high-importance level of link *l* in network *g=(N,L)*, denoted as *HIL(g)*, is described as follows:

$$HIL(g) = \{l \mid l \in L, IMPL(l) = 2\} \quad (3)$$

Therefore, *HIL(g)* is the set of links that the value of *IMPL* of link *l* is 2, which means the link *l* has the highest importance level so that the flows going through it has the requirements of lower latency and higher transmission quality. Thus we apply double-path backup strategy to the link *l*, which provides two backup paths that can recover link failure from the detectable-switch to the next-switch (denoted as NHOP) and the next-next-switch (denoted as NNHOP) belonging to the working path, and the NHOP backup path is the prior one. The strategy can ensure that if the link *l* fails and it is unable to recover normal communication from the prior backup path, the flows through the link *l* can be transferred to the NNHOP backup path.

*2) Single-Path Strategy:* The middle-importance level of link *l* in network *g=(N,L)*, denoted as *MIL(g)*, is described as follows:

$$MIL(g) = \{l \mid l \in L, IMPL(l) = 1\} \quad (4)$$

Where the *MIL(g)* is the set of links that the value of *IMPL* of link *l* is 1, which means the link *l* has the middle importance level so that the flows going through it has lower demand than the flows through high-importance level links. Thus we apply single-path backup strategy to the link *l*, and we just pre-configure the NHOP backup path to recover the link failure so that the required failover delay can be satisfied. If this strategy fails, we trigger the reactive recovery strategy.

*3) Reactive Strategy:* The low-importance level of link *l* in network *g=(N, L)*, denoted as *LIL(g)*, is described as follows:

$$LIL(g) = \{l \mid l \in L, IMPL(l) = 0\} \quad (5)$$

Thus the *LIL(g)* is the set of links that the value of *IMPL* of link *l* is 0, which means the number of flows going through link *l* is fewer or those flows have no serious demand for the latency or packet loss, to which we apply reactive backup strategy. In this strategy, backup path is dynamically allocated, but resources required by recovery paths are not allocated until a failure occurs. Thus, when the failure occurs, additional signaling is required to establish the backup path.

## C. Backup-Resource Based Approach

Given a network *G= (N, L)* and *IMPL*, we can construct sets with different level, such as *HIL*, *MIL* and *LIL*, by the Algorithm1, where each $l \in L$ will be classified by the value of *IMPL*.

**Algorithm1** Classify links

```
Input: G= (N, L), IMPL
Output: HIL, MIL, LIL
Procedure:
 1: for l in L
 2:   if IMPL(l)==2
 3:       HIL=l
 4:       else if IMPL(l)==1
 5:           MIL=l
 6:       else LIL=l
 7:       end if
 8:   end if
 9: end for
```

We use Algorithm2 to compute the backup paths for links, the backup paths are computed by the strategy proposed in IV.B according to the *IMPL*. Specifically, if more than one appropriate paths are available, we will select the one according to hop, bandwidth and *BPN* in sequence. Moreover, the variable *K* used in NeworkX [16] can also be configured according to the actual situation.

**Algorithm2** Compute backup paths for links

```
Input: G= (N, L), IMPL, K, HIL
Output: L.BestPath, L.SecondPath
Procedure:
 1: for l in L
 2:   Select K paths with the minimum number of hop
        for link l as NHOP backup path using NetworkX,
        and stored in KPath
 3:   if KPath[0].hop==KPath[1].hop
 4:       Sort KPath by ascending order in bandwidth
 5:       else if KPath[0]. bandwidth==
                         KPath[1].bandwidth
 6:           Sort KPath by ascending order in BPN
 7:           L[l].BestPath= KPath[0]
 8:       end if
 9:   end if
10:   if l in HIL
11:     NLink=get the remaining links connecting with the
           end node of l
12:     for nl in NLink
13:       L[l].SecondPath=select the best
14:       NNHOP backup path from the source node of l
              to the end node of nl with the same
              compute priority as the NHOP
15:     end for
16:   end if
17: end for
```

After computing the backup paths for links by Algrithom1 and Algorithm2, controller sends the flow entries for failure recovery to the related switches. In addition, with the network operating, the importance level of links will vary, thus the controller monitor the *FN* and *BUR* once every time *T*, in our work we set *T* as 5ms, and if the real time value of *IMPL* is different from older value, then update the backup paths.

## V. PERFORMANCE EVALUATION

In this section, we conduct simulations to verify our approach. We let *Nb* present the total number of flow entries for backup paths, and let *Nw* present the total number of flow entries for working paths. The value of *Nb* varies with different recovery approaches, thus we first compare our approach with two existing approaches mainly in terms of the value of *Pb = Nb/(Nb+Nw)*. Then we evaluate the performance of our approach with different scales and topologies.

## A. Comparison with Existing Approaches

In order to verify the validity of our approach, denoted as IML, we compare it with two typical existing approaches: the approach proposed by Capone, A et al. (denoted as DP) [7] and the approach proposed by Adrichem, Niels L. M. Van et al. (denoted as CR) [6], which are described in TABLE I. Moreover, we first select the German Backbone Network Topology from SNDlib [17], which has 14 switches and 21links. And we divide the switches in two sets: edge switches and core switches. Edge switches act as source and destination of flows while core switches are only in charge of routing. We assume that the failure probability of each link is same, and the biggest bandwidth of each link is 1024M.

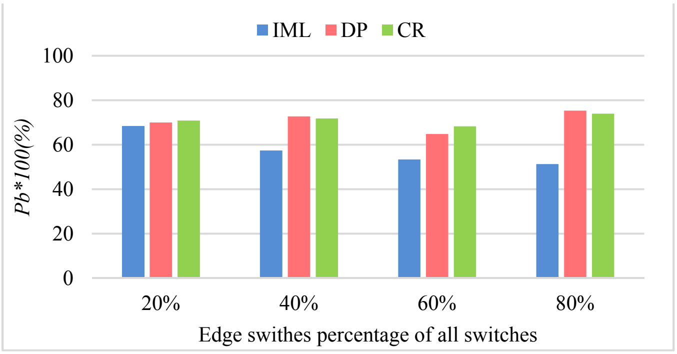

Fig.3. The value of *Pb* with different number of edge switches

TABLE I. THEREE FAILURE RECOVERY APPROACHES

| Approach | Failure Recovery Strategy |
|---|---|
| IML | Physical links is graded to three kinds of levels, and each link uses different backup strategy that we proposed. |
| DP | The "*crankback*" routing strategy are used when link failure [7]. |
| CR | The backup path is calculated from every intermediate switch to destination in case of link-failure, and primary and secondary paths are preconfigured [6]. |

In order to assess the performance of resource consumption for configuring the backup paths, we first set the percentage of edge switches as 20%, 40%, 60% and 80%, and we collect the data of *Nb* and *Nw*, then we can get the value of *Pb* with different percentage of edge switches. As we can see from the Fig.3, the value of *Pb* in DP and CR is between 60% and 75%, which is greater than IML. Moreover, with the number of edge switches increasing, the value of *Pb* is basically flat both in DP and CR. However, our approach IML gradually levels off after the value of *Pb* decreasing, because IML only consider the physical link, and the backup paths are independent of flows. Therefore, even though the number of backup flow entries varies with the number of flows augment, the value of *Pb* reduces in comparison with DP and CR.

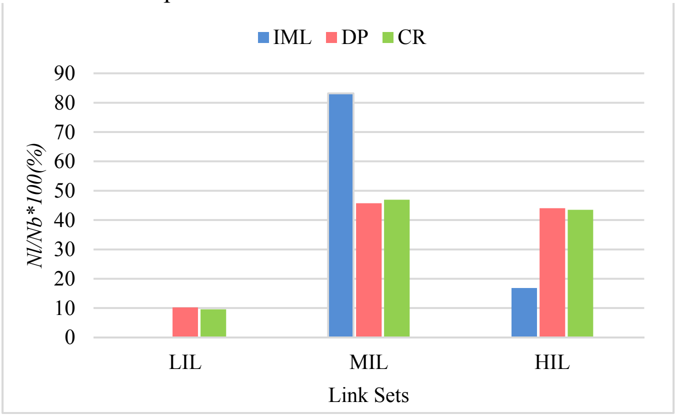

Fig.4. The value of *Nl/Nb* with different link sets

In order to test the number of flow entries in each set accounted for the proportion of the flow entries for backup paths, we select 40% of all switches in the topology as edge switches, and set the coefficient $\alpha$ as 0.4, then we observe how many flow entries each approach will pre-configure for failure recovery. We gather the stored flow entries for backup in the set of *HIL*, *MIL* and *LIL* respectively, and we let *Nl* represent the total number of flow entries in one of the sets. The result is shown in Fig.4, we can find that the value of *Nl*/*Nb* in CR and DP is of about the same at each kind of set, since they do not class the links, which cannot take different recovery strategies to links belonging to different sets. And whether the backup paths for link is pre-configured or not is largely up to flows going through the link, thus the value of *Nl/Nb* in the set of *LIL* in CR and DP is lower than other sets. Moreover, the controller does not need to pre-configure backup paths for the links in the set of *LIL*, thus in IML, the value of *Nl*/*Nb* in the set of LIL is zero.

Then we evaluate the three approaches in different networks, we use BRITE [18] to randomly generate networks with the number of switches from 10 to 190. In Fig.5, we set $\alpha$ as 0.4 and the connecting probability between any two switches as 50%, and compare IML with other approaches. We can find that when the number of switches increases, the value of *Pb* almost stays at around 60% and has no large floating in DP and CR. However, it gradually reduces in IML, and the overall value lower than other approaches. The reason is that with the scale of network increasing, the flows will increase to satisfy greater business volume, then the number of flows going through the same link will augment, which is a good news for IML, because IML just compute backup path once for a physical link, thus it will save more storage resources of flow table in this case.

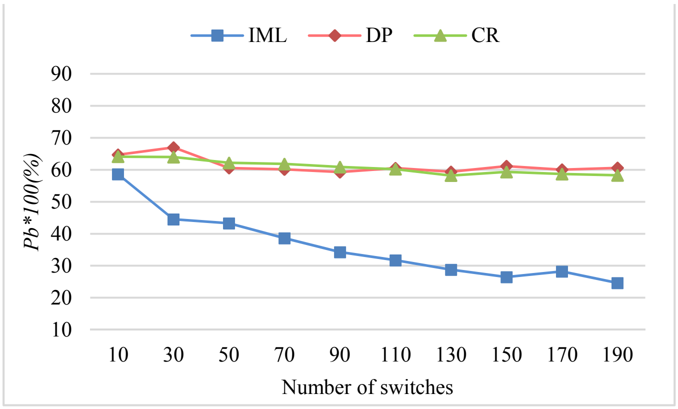

Fig.5. The value of *Pb* in three approaches with different number of switches

## B. *Performance in Different Networks*

We also evaluate our approach in networks with different scales. In Fig.6 and Fig.7, we set the edge switches as 40%.

Firstly, we set the value of α as 0.2, 0.4, 0.6 and 0.8 to observe the number of flow entries for pre-configuring backup paths in our approach. We can find in Fig.6 that with the number of switches increasing, all values of *Pb* decrease in different α. Because with the number of switches and business volume increasing, the number of flows going through the same physical link will augment, and the backup paths for the physical link failure recovery only compute once, thus the flow entries for the link recovery will reduce a lot. Moreover, when the number of switches is more than 70, the value of *Pb* keeps lowest in α=0.4 than others.

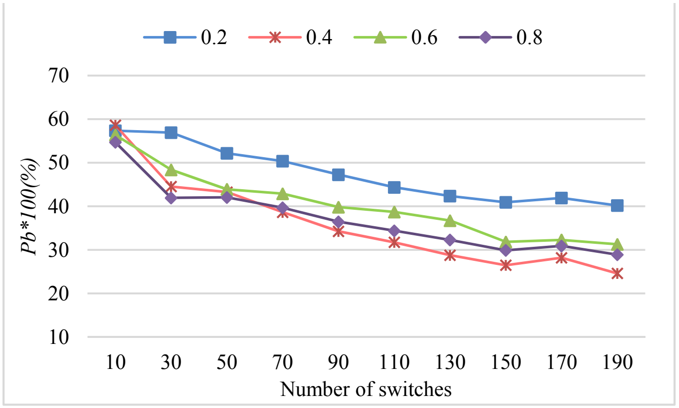

Fig.6. The value of *Pb* with different value of α

In order to observe the influence on the value of *Pb* with different connecting probability between any two switches, we set the connecting probability between any two switches as 40%, 60% and 80%. The result is shown in Fig.7, we can find that as the connecting probability gets smaller, more flow entries are needed to pre-configure the backup paths. Because when the connecting probability gets smaller, the average hops between any two switches may get bigger, more flow entries are needed to recover the link failure. In addition, no matter how big the connecting probability is, the value of *Pb* reduces with the scale of network increasing, which means our approach has good performance for resources consumption in networks of different scales and connectivity.

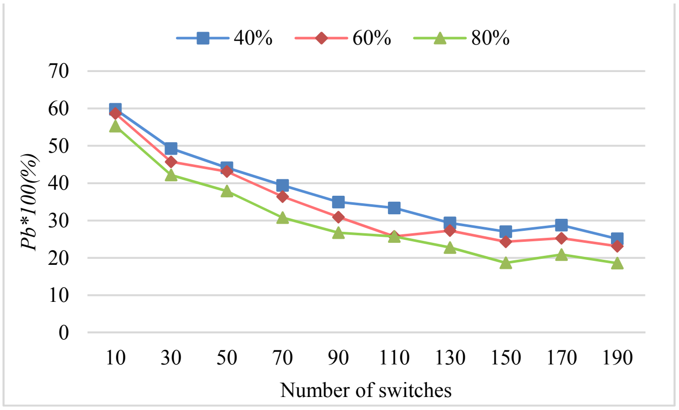

Fig.7. The value of *Pb* with different connecting probility

## VI. CONCLUSION

In this paper, we have investigated the single link failure recovery problem in SDN. The deficiencies of existing approaches are summarized. In order to minimize the flow entries for backup paths and meet the required failure recovery delay, we propose a backup resource-based failure recovery approach. We first propose two metrics to determine the importance level of a link. Then we propose three kinds of strategies for different graded links, based on which we give the formulation of the problem. We also design algorithms to find adaptive backup paths for links with less flow entries. Simulations show that our approach can reduce flow entries for link failure recovery, and meet the required delay of the important traffic at the same time, and has a good performance in networks of different scales and connectivity.

## ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (61501044).

## REFERENCES

[1] https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf.

[2] Sharma, S.; Staessens, D.; Colle, D.; Pickavet, M.; Demeester, P., "Enabling fast failure recovery in OpenFlow networks," in Design of Reliable Communication Networks (DRCN), 2011 8th International Workshop on the , vol., no., pp.164-171, 10-12 Oct. 2011.

[3] Sharma, S.; Staessens, D.; Colle, D.; Pickavet, M.; Demeester, P., "Fast failure recovery for in-band OpenFlow networks," in Design of Reliable Communication Networks (DRCN), 2013 9th International Conference on the , vol., no., pp.52-59, 4-7 March 2013.

[4] Staessens, D., et al. "Software defined networking: Meeting carrier grade requirements." IEEE Workshop on Local & Metropolitan Area NetworksIEEE, 2011:1-6.

[5] Sharma, Sachin, et al. "OpenFlow: Meeting carrier-grade recovery requirements." Computer Communications 36.6(2013):656-665.

[6] Adrichem, Niels L. M. Van, B. J. V. Asten, and F. A. Kuipers. "Fast Recovery in Software-Defined Networks." 2014 Third European Workshop on Software Defined Networks (EWSDN) IEEE Computer Society, 2014:61-66.

[7] Capone, A., et al. "Detour planning for fast and reliable failure recovery in SDN with OpenState." Design of Reliable Communication Networks (DRCN), 2015 11th International Conference on the IEEE, 2015:25-32.

[8] Kempf, J.; Bellagamba, E.; Kern, A.; Jocha, D.; Takacs, A.; Skoldstrom, P., "Scalable fault management for OpenFlow," in Communications (ICC), 2012 IEEE International Conference on , vol., no., pp.6606-6610, 10-15 June 2012.

[9] Adami, D.; Giordano, S.; Pagano, M.; Santinelli, N., "Class-based traffic recovery with load balancing in software-defined networks," in Globecom Workshops (GC Wkshps), 2014 , vol., no., pp.161-165, 8-12 Dec. 2014.

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

[11] Stephens B, Cox A, Felter W, Dixon C, Carter J. "PAST: scalable ethernet for data centers." ACM SIGCOMM CC 2012: 49-60. DOI: 10.1145/2413176.2413183.

[12] Kanizo, Y., D. Hay, and I. Keslassy. "Palette: Distributing Tables in Software-Defined Networks." Proceedings - IEEE INFOCOM 12.11(2013):545-549.

[13] Zhang, Xin, et al. "DPPC-RE: TCAM-Based distributed parallel packet classification with range encoding." IEEE Transactions on Computers 55.8(2006):947-961.

[14] Ma, Huan, Y. Yang, and Z. Mi. "A distributed storage framework of FlowTable in software defined network." Computers & Electrical Engineering 43(2015):155-168.

[15] V. Padma and P. Yogesh, "Proactive failure recovery in OpenFlow based Software Defined Networks," Signal Processing, Communication and Networking (ICSCN), 2015 3rd International Conference on, Chennai, 2015, pp. 1-6.

[16] http://networkx.github.io/.

[17] http://sndlib.zib.de.

[18] http://www.cs.bu.edu/brite/.