# A Customized and Cost-Efficient Backup Scheme in Software-Defined Networks

Zhijie Zhu*‡, Qing Li†, Mingwei Xu*, Ziyan Song*‡ and Shutao Xia‡
*Department of Computer Science and Technology, Tsinghua University, Beijing, China
†Network Technology Laboratory, Huawei Technologies Co.Ltd, Hong Kong
‡Graduate School at Shenzhen, Tsinghua University, Shenzhen, China
{zhuzj15, szy15}@mails.tsinghua.edu.cn; liqing53@huawei.com; xumw@tsinghua.edu.cn; xiast@sz.tsinghua.edu.cn

***Abstract*—Among the schemes proposed for failure recovery in software-defined networks, installing backup paths in advance is considered to be an effective approach to reduce the recovery latency. However, the pre-installation poses undue storage overheads on flow tables. In this paper, we propose a customized and cost-efficient backup scheme, which achieves fast recovery from any single-link failure. We introduce an improved breadth first search algorithm to construct the customized backup paths of flows to accommodate their diverse routing demands. By analyzing the path characteristics carefully, we observe that non-conflicted backup paths can be aggregated with each other to reduce the number of backup rules. We formulate this backup path aggregation as an optimization problem. The challenge is how to aggregate the backup paths without causing routing ambiguity. To this end, we design a two-stage aggregation (2SA) algorithm. At its core, 2SA leverages conflict matrixes to guarantee the correctness of aggregation. We evaluate our scheme comprehensively with both the real-world topologies and generated topologies. Simulation results show that our scheme can pre-install the customized backup paths with much fewer, typically one order of magnitude, backup rules compared to the traditional flow-based protection without aggregation.**

## I. Introduction

Software-Defined Networking (SDN) is a new network architecture, which decouples the control logic from the data forwarding operations, so as to simplify network management and enable more flexible network control. In SDN, a logically centralized controller determines how forwarding devices behave by directly programming their flow tables via OpenFlow protocol [1]. These new characteristics of SDN benefit some complicated tasks, such as globally optimal traffic engineering and security applications on commodity switches [2][3].

SDN brings about opportunities as well as challenges for network failure recovery. Basically, most of the recovery schemes proposed in recent years adopt either of the following policies: failure restoration and failure protection [4]. Failure restoration is reactive, where the time to restore a broken path, besides the detection time, includes the delay of communication between the affected switches and the controller, path re-computation, and re-configuration of the network. As a result, this controller-initiated restoration may spend over 100 ms [5] to finish, which is intolerable for latency-sensitive flows like Voice-over-IP and stock exchange data services.

In contrast, by pre-configuring backup paths, failure protection enables the recovery to proceed totally in the data plane and thus minimizes the recovery time to the delay of failure-detection. Through these pre-configured backup paths, affected traffic can be immediately rerouted around the failed component, without involving the controller. Some protection schemes [6][7] have been proposed. These schemes configure multiple backup paths for each flow a priori to handle the potential failures of the links the flow traverses. Though such flow-based protection is flexible in calculating backup paths, massive backup rules are needed to install, which is a big challenge for the scarce flow table memory [5].

In this paper, we propose a flow-based backup scheme in SDN, which achieves backup path customization, high cost-effectiveness, and fast recovery. Our scheme first constructs the customized backup paths for all flows to accommodate their potential specific routing demands in the real scenarios. Specifically, we relate each flow with two demand sets (*pass* and *avoid*) and design an improved breadth first search (IBFS) algorithm which leverages the demand sets to achieve backup path customization. To reduce the number of backup rules, our scheme aggregates these backup paths without causing routing ambiguity by our developed two-stage aggregation (2SA) algorithm. 2SA achieves a significant reduction in backup rules with a relatively low latency overhead. Finally, our scheme pre-installs all the customized backup paths with only a small number of backup rules. Through these pre-installed backup paths, our scheme can reroute the affected flows immediately once any failure occurs, so as to guarantee fast recovery.

To demonstrate the performance of our scheme, we construct comprehensive experiments with the real-world topologies from Rocketfuel [8] and the generated topologies of BRITE [9]. The experiment results show that our scheme requires much fewer, typically one order of magnitude, backup rules than that of the flow-based protection without aggregation. It also shows that our scheme does not pose a heavy burden on the controller as it can always terminate in several microseconds for small topologies and dozens of seconds for large topologies.

## II. Scheme Overview

### A. Customized Backup Paths

To achieve the diverse business requirements (such as service quality and safety), some routing demands of flows should be guaranteed, even when failures occur. Thus, a desirable backup path should not only detour the affected traffic around the failed component but also guarantee these routing demands. We refer to such a desirable backup path as *a customized backup path*. At the first step, our scheme focuses on constructing these customized backup paths.

## *B. System Framework*

Fig. 1 overviews our backup scheme. The implementation consists of two main modules: backup path construction and backup path aggregation. Besides, our scheme receives diverse routing demands from the application layer as the input.

- *Backup path construction:* This module aims to construct the customized backup paths for all flows. At the first, we need to translate the high-level routing demands to the corresponding low-level forwarding decisions. To this end, we relate each flow with two demand sets, *pass* and *avoid*, to reflect its routing demands. Specifically, the *pass* set indicates which node(s) the flow should traverse, while the *avoid* set specifies the node(s) that the flow should avoid. To construct the backup paths for a flow with its related demand sets followed, we propose an improved breadth first search (IBFS) algorithm. We detail it in Section III.
- *Backup path aggregation:* This module is the heart of our backup scheme. The problem is how to aggregate the massive customized backup paths without causing any routing ambiguity so that the storage cost on flow tables is minimized. We formulate it as an optimization problem to minimize the number of backup rules. To solve it efficiently, we design a two-stage aggregation (2SA) algorithm consisting of per-flow aggregation and cross-flow aggregation. 2SA guarantees routing unambiguity and achieves high cost-effectiveness. We elaborate the formulation in Section IV and the 2SA algorithm in Section V.

We note that simultaneous link failures may happen in real networks. Fortunately, if we consider the network convergence time in the 10-second scale, which has been demonstrated as a conservative number, the number of two simultaneous failures is almost zero and the probability of three or more simultaneous failures is negligible [10]. Thus, we design our backup scheme focusing on single link failures.

## III. Construction of Customized Backup Paths

For each flow, we need to construct a group of customized backup paths to protect each link on its working path. To this end, we design an improved breadth first search (IBFS) algorithm. IBFS leverages the demand sets (*pass* and *avoid*) to guarantee the routing demands of each flow. Besides, the backup paths constructed by IBFS tend to pass through fewer hops and thus require fewer backup rules. IBFS operates as follows:

- *Starting from the upstream node of the failed link, IBFS attempts to hunt for a path to any subsequent node in the flow's working path, without traversing the failed link and any node in the* avoid *set.*
- *If all the nodes in the* pass *set have been contained in either the searched path or the remaining part of the working path, then the searched path is an applicable customized backup path.*
- *Otherwise, IBFS continues to search another subsequent node along the working path in reverse order and check if the newly searched path is applicable.* Since the working path implicitly achieves the routing demands, by backtracking the working path, IBFS can certainly find an applicable customized backup path.

IBFS repeats the above operations and finally returns the customized backup paths (if any) for all flows.

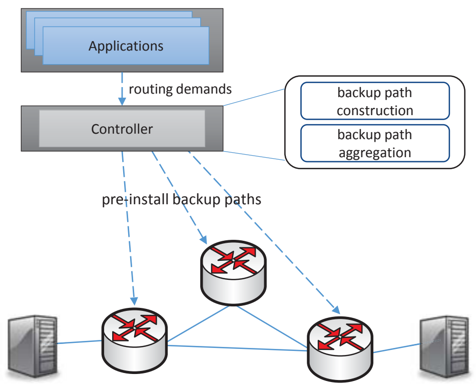


Fig. 1. System framework of our backup scheme.

## IV. Aggregation of Backup Paths

## *A. Problem Description*

Usually, each backup path is implemented as a tunnel identified by a unique label, like MPLS and VLAN tags. Though this property simplifies the configuration of backup paths, it requires a large number of backup rules to install, which imposes a prohibitive overhead on the flow tables. Thus, we propose to aggregate the backup paths that can share the same label without causing routing ambiguity, so as to reduce the number of backup rules. Then, what kinds of backup paths can be aggregated? Without loss of generality, two backup paths have three basic relations between each other, i.e., non-conflicted, disjoint, and conflicted as shown in Fig. 2. We define that two backup paths are non-conflicted if they have a common end node and only one convergent sub-path (i.e., the segment from their first joint node to the same end node). It is straightforward that non-conflicted backup paths can be aggregated, while conflicted backup paths can not. Besides, disjoint backup paths can also be aggregated. However, aggregating disjoint backup paths is helpless to reduce backup rule number. Thus, aggregation should be performed only on non-conflicted backup paths.

Fig. 3 illustrates the main idea of the aggregation of non-conflicted backup paths. In this figure, *p2* and *p3* are two non-conflicted backup paths of the flow *h1*–*h2*. To aggregate them, our scheme unifies their labels to *t1*. As a result, on their common sub-path *A*–*C*–*D*, our scheme can configure just one backup rule in each node to save the overhead on flow tables.

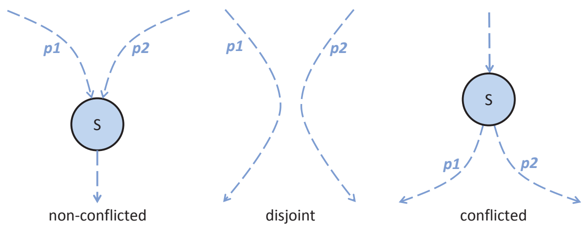


Fig. 2. Three basic relations between two backup paths.

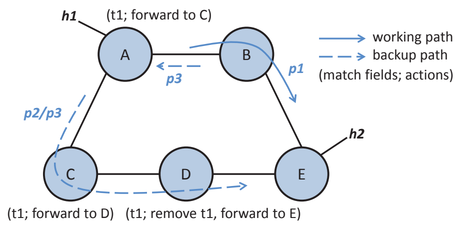


Fig. 3. Example demonstrating the aggregation of backup paths. The fast-failover group entries in *A* and *B* which are used to insert the label *t1* to the affected packets, are omitted.

TABLE I
NOTATIONS FOR OPTIMIZATION

| Notation | Meaning |
|---|---|
| $(\mathbf{V}, \mathbf{E})$ | A network with the node set $\mathbf{V}$ and the link set $\mathbf{E}$. |
| $\mathbf{F}$ | The set of flows. |
| $f_{s,d}$ | A flow in $\mathbf{F}$, from source $s$ to destination $d$. |
| $\mathbf{BP}$ | The list of backup paths. |
| $bp_i$ | The *i-th* backup path in $\mathbf{BP}$, where $i \in [1, n]$. |
| $n$ | The number of backup paths in $\mathbf{BP}$. |
| $l_i$ | The number of backup rules that are needed to configure $bp_i$ individually. |
| $\delta^e_{s,d}$ | Binary, $\delta^e_{s,d} = 1$ if the link $e$ is on the working path of $f_{s,d}$; 0, otherwise. |
| $\delta^{e,i}_{s,d}$ | Binary, $\delta^{e,i}_{s,d} = 1$ if $bp_i$ is constructed to protect the flow $f_{s,d}$ from the failure of link $e$; 0, otherwise. |
| $\alpha_{i,j}$ | Binary, $\alpha_{i,j} = 1$ if $bp_i$ and $bp_j$ are different backup paths and non-conflicted; 0, otherwise. |
| $g_{i,j}$ | The reduced number of backup rules by aggregating $bp_i$ and $bp_j$; $g_{i,j} = 0$ if $bp_i$ and $bp_j$ are conflicted or disjoint. |
| $x_{i,j}$ | Binary, $x_{i,j} = 1$ if $bp_i$ and $bp_j$ are aggregated; 0, otherwise. |
|  | $e \in \mathbf{E}$. |

Finally, the general problem is that given the set of customized backup paths, how to divide it into disjoint subsets so that any two backup paths in the same subset are non-conflicted and the storage cost for backup rules after aggregation on each subset is minimized. We first formulate this problem in Section IV-B and then design an efficient algorithm to solve it in Section V.

## *B. Formulation*

To figure out the number of backup rules that are required to pre-install after the aggregation, we formulate the backup path aggregation as an optimization problem. The notations used in the formulation are summarized in Table I. The basic requirements for this problem are: 1) for each flow and each link on its working path, there should be one applicable backup path pre-configured, 2) any two backup paths which are aggregated should be non-conflicted, and 3) all the backup paths that have been aggregated with the same backup path should be non-conflicted with each other. This leads to the following formulation:

**Minimize:**

$$\sum_{1 \le i \le n} l_i - \sum_{1 \le i < n} \max_{i < j \le n} \{x_{i,j} g_{i,j}\} \tag{1}$$

**Subject to:**

$$\delta^e_{s,d} \le \textstyle\sum_{i=1}^{n} \delta^{e,i}_{s,d}, \forall f_{s,d} \in \mathbf{F}, e \in \mathbf{E} \tag{2}$$

$$x_{i,j} \le \alpha_{i,j}, \forall i, j \in [1, n] \tag{3}$$

$$x_{i,j} = 1, \forall i, j, k \in [1, n], i \ne j, x_{i,k} = x_{j,k} = 1 \tag{4}$$

In the above formulation, (2) ensures that any link $e$ on the flow $f_{s,d}$ should be protected by an applicable backup path $bp_i$. (3) requires that two backup paths $bp_i$ and $bp_j$ are allowed to be aggregated only when they are non-conflicted. (4) ensures that any two different backup paths $bp_i$ and $bp_j$ should be aggregated if both of them have been aggregated with another common backup path $bp_k$. The objective that minimizes the number of backup rules is described in (1), where $\sum_{1 \le i \le n} l_i$ represents the number of backup rules when configuring the backup paths independently, while $\sum_{1 \le i < n} \max_{i < j \le n}\{x_{i,j} g_{i,j}\}$ refers to the number of backup rules that can be saved through aggregation.

By solving the above problem, we know the disjoint non-conflicted sets of backup paths and the minimal number of backup rules which are required to install.

## V. Two-Stage Aggregation Algorithm

The above optimization is a typical 0-1 integer linear programming problem, which is well-known as an NP-complete problem [11]. Therefore, we design the following two-stage aggregation (2SA) algorithm to solve this problem in polynomial time. 2SA achieves the aggregation of backup paths without causing routing ambiguity. More specifically, 2SA consists of two stages: 1) per-flow aggregation, and 2) cross-flow aggregation.

## *A. Per-Flow Aggregation (Stage I)*

For the per-flow aggregation (PFA), the goal is to aggregate the non-conflicted backup paths belonging to the same flow. In this stage, we start from the group of backup paths constructed by IBFS. We design the PFA algorithm whose main idea is as follows.

- *First, we sort the multiple backup paths of each flow into an increasing sequence.* We say a backup path has a higher sequence number than the other if its end point (on the flow's working path) is nearer to the destination. In the case that two backup paths have the same end point, their sequence is determined alike by their start points.
- *Second, for the sorted backup path sequence of the same flow, we check each backup path in turns as the inverse*

*order and attempt to aggregate it with its next neighbor or neighbors (if its next neighbor has been aggregated with its next-next neighbor.* To this end, we try to re-construct a new backup path, which can be aggregated with its subsequent neighbor(s), and examine whether it is applicable (i.e., bypassing the failed link and guaranteeing the associated routing demands) and requires less backup rules after aggregation than the original one without aggregation. If the answer is yes, we replace the original backup path with this newly constructed one and aggregate the new one with its next neighbor(s). Otherwise, we reject the reconstructed backup path and continue to check the sequence. The algorithm terminates after we examine all sequences and finally returns the aggregated backup paths of each flow.

## *B. Cross-Flow Aggregation (Stage II)*

In Stage I, we reduce the number of backup rules by aggregating the non-conflicted backup paths belonging to the same flow. As a result, the backup paths for each flow are aggregated into multiple groups, and all the paths in one group are identified with the same path label. In form, a group looks like a reverse tree, any path of which from a leaf node to the root (i.e., the common end point) represents a specific backup path. We call such groups *BP-tree*s and define that two BP-trees are *homogeneous* if they have the same root and the same child node of the root, as shown in Fig. 4.

For the cross-flow aggregation (CFA), the goal is via aggregating the homogeneous BP-trees to further lessen the backup rules. Similarly, we assert that two homogeneous BP-trees are non-conflicted only if any two backup paths from each of them are non-conflicted. Obviously, only when two homogeneous BP-trees are non-conflicted, it is legitimate to aggregate them.

To guarantee the legitimation of BP-tree aggregation, we use a matrix $M$ to describe the confliction relation between any two BP-trees in a homogeneous BP-tree group. Suppose a group has $T$ homogeneous BP-trees, whose labels are encoded as $t_i$ $(1 \le i \le T)$ and differ from each other. We then use $m_{ij}$ $(1 \le i, j \le T)$ to indicate whether BP-trees $t_i$ and $t_j$ are conflicted. If $m_{ij} > 0$, it means $t_i$ and $t_j$ are non-conflicted and the *gain* (the reduced number of backup rules by aggregating them) is $m_{ij}$, and 0 otherwise means $t_i$ and $t_j$ are conflicted.

$$M(t_1, t_2, \cdots, t_T) = \begin{array}{c} \\ t_1 \\ t_2 \\ \vdots \\ t_T \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & \cdots & t_T \end{array} \\ \begin{pmatrix} m_{11} & m_{12} & \cdots & m_{1T} \\ m_{21} & m_{22} & \cdots & m_{2T} \\ \vdots & \vdots & \ddots & \vdots \\ m_{T1} & m_{T2} & \cdots & m_{TT} \end{pmatrix} \end{array}$$

Based on the above thoughts, we propose CFA algorithm whose main idea is as follows.

- *First, we divide all homogeneous BP-trees into a group, so as to acquire the set of all homogeneous groups.*
- *Second, we compute the initial conflict matrix M for each homogeneous group.* For two distinct BP-trees in a group, such as $t_i$ and $t_j$, we judge whether they are conflicted or not by examining their common nodes. Specifically, if there exists one common node (apart from the root node which has no parent) whose parents in $t_i$ and $t_j$ are different, then $t_i$ and $t_j$ are conflicted because of the routing ambiguity appearing on this common node. Otherwise, they are non-conflicted and their gain by aggregation equals to the number of their common nodes, excluding the root and the common leaf nodes (if any), where no backup rules are needed to install.
- *Last, on the basis of M, we use a greedy approach to aggregate the BP-trees in each homogeneous group.* Each time, we choose the two BP-trees associated with the maximum gain and merge them to a larger BP-tree by unifying their path labels. Then we update $M$ to reflect the new conflict relations between the newly-merged BP-tree and the remaining ones. More specifically, if either of the two original BP-trees is conflicted with another BP-tree, then the merged BP-tree should be also conflicted with it; otherwise, the new BP-tree should still be able to aggregate with it and the gain should equal to the larger one acquired by individually merging the two old BP-trees with it. We repeat this *choose-merge-update* process until there is no positive element in *M*. This means all non-conflicted BP-trees have been aggregated. Finally, the algorithm terminates after it completes the BP-tree aggregation for all homogeneous groups and returns the final disjoint non-conflicted sets of all backup paths.

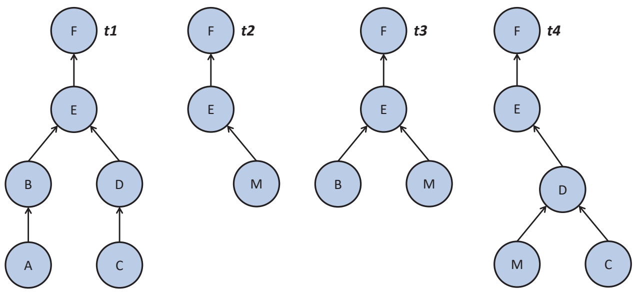


Fig. 4. Four homogeneous BP-trees and their path labels.

In Fig. 5 we use a simple example to walk readers through the algorithm. Assume a whole homogeneous group consists of the four BP-trees shown in Fig. 4. We first compute the conflict matrix by counting the common nodes which share the same parent between any two BP-trees in the group. For example, as each non-root common node $(C, D, or\ E)$ between $t_1$ and $t_4$ has the same parent, and both $D$ and $E$ are not leaf nodes, we have $m_{14} = m_{41} = 2$. In contrast, as the common node $M$ of $t_3$ and $t_4$ has two distinct parents, $E$ in *t3* and $D$ in *t4*, we know that $t_3$ and $t_4$ are conflicted and have $m_{34} = m_{43} = 0$. Accordingly, we can acquire the initial conflict matrix $M_0$ of the four BP-trees. Next, we find out the maximum gain, $m_{14}$ $(= 2)$, and merge the two BP-trees, $t_1, t_4$, related by it, via changing the path label of $t_1$ to that of $t_4$. After that, we update $M_0$ to reflect the new conflict relations between $t_2, t_3$ and the newly-aggregated $t'_4$. Since $t_4$ is conflicted with both $t_2$ and $t_3$ $(m_{42} = m_{43} = 2, in\ M_0)$, $t'_4$ should keep

$$M_0(t_1,t_2,t_3,t_4) = \begin{bmatrix} 0 & 1 & 1 & \underline{2} \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{t_1 \mapsto t_4} M_1(t_2,t_3,t'_4) = \begin{bmatrix} 0 & \underline{1} & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \xrightarrow{t_2 \mapsto t_3} M_2(t'_3,t'_4) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Fig. 5. Walk-through example on CFA algorithm: each underlined positive element in $M_k$ $(0 \leq k \leq 1)$ is a maximum gain, indicating that the two BP-trees it relates should be aggregated, and $t_x \mapsto t_y$ on each labeled arrow means aggregating $t_x$ to $t_y$.

conflicted with them too. Thus we acquire $M_1$. We perform the same operation for the subsequent conflict matrixes one by one and finally acquire $M_2$, where there is no positive elements any more. Therefore, the final BP-trees after the cross-flow aggregation are $t'_3 = \{t_2, t_3\}$ and $t'_4 = \{t_1, t_4\}$, and the total gain is $2 + 1 = 3$.

## VI. Evaluation

### A. Set Up

We have implemented our backup scheme as a Ryu application [12]. To simulate the routing demands of flows, we randomly select about $10\%$ flows and specify them to pass through certain nodes, and meanwhile randomly extract about $10\%$ flows and force them to avoid specific nodes, regardless of whether a single-link failure happens. For the remaining flows, we compute their working paths by the shortest path first (SPF) algorithm as the default. For simplicity, we define a flow at the granularity of source to destination in our evaluation. We state a flow is *new* when it is set up the fist time or when its associated routing demands vary.

All the simulations are conducted on an Ubuntu 16.04 laptop with 2.2GHz Intel CPU and 8GB memory.

### B. Number of Backup Rules

To investigate how many backup rules are needed to guarantee the network availability in the presence of any single-link failure, we apply our proposed backup scheme to dozens of real topologies [8] to protect all their flows. The major evaluation criterium we use is *aggregation rate*, i.e., the ratio of the saved number of backup rules by our scheme to the number required by traditional flow-based protection (without aggregation). The result is summarized in Table II. From the table, we observe that for a specific topology, our scheme requires much fewer, typically one order of magnitude, backup rules compared to the traditional flow-based protection.

Fig. 6(a) shows the more detailed results containing also the backup rule numbers by link-based protection, on three example topologies whose basic profiles are listed in Table III. We see that even in contrast to the link-based protection, which is usually less costly than the flow-based protection (however, it lacks flexibility), our scheme is more efficient. This suggests that our scheme achieves high cost-effectiveness and we can easily fit these backup rules into today's SDN switches which typically have few thousands of flow entries available. In addition, it also indicates that in a larger network, our approach can achieve higher aggregation rate, e.g., 88.37% for AS6509 vs. 97.55% for AS701.

TABLE II
Overview of aggregation rates.

| Dataset | Aggregation Rate | Dataset | Aggregation Rate |
|---|---|---|---|
| AS6539 | 0.6333 | AS6939 | 0.75 |
| AS3602 | 0.825 | AS2497 | 0.8175 |
| AS6509 | 0.8837 | AS11537 | 0.8789 |
| AS6461 | 0.9081 | AS6467 | 0.8825 |
| AS7911 | 0.9251 | AS8220 | 0.9267 |
| AS6395 | 0.9124 | AS4637 | 0.9015 |
| AS5400 | 0.9605 | AS3257 | 0.9594 |
| AS4323 | 0.9518 | AS1239 | 0.9543 |
| AS209 | 0.9668 | AS701 | 0.9755 |

TABLE III
Example topology profiles.

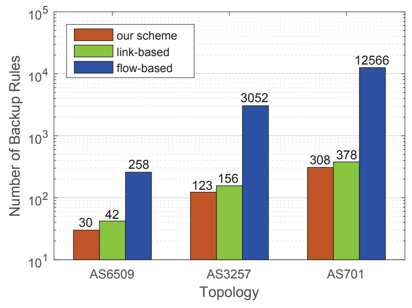
| Dataset | Number of Nodes | Number of Links | Degree |
|---|---|---|---|
| AS6509 | 12 | 22 | 3.67 |
| AS3257 | 41 | 87 | 4.24 |
| AS701 | 83 | 219 | 5.28 |

To further analyze the burden on each switch, Fig. 6(b) shows the scatter graph of the average number of backup rules in each switch in AS701. We can see that, with our scheme, all switches are configured with only a few or dozens of backup rules, while with the traditional flow-based scheme, most switches are configured with hundreds of backup rules. What is worse, there are a few switches allocated thousands of backup rules, which is intolerable.

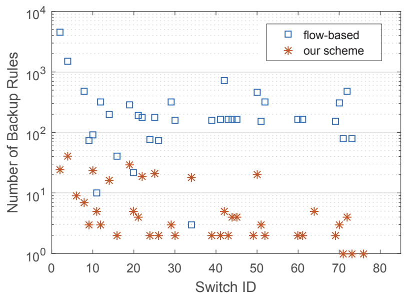
In addition, to investigate how the numbers of nodes and links of a network influence the number of backup rules, we use BRITE to systematically synthesize topologies with different numbers of nodes and links. As shown in Fig. 6(c), we can observe that the aggregation rate of the topologies with a constant degree increases along with the growth of the number of nodes. The result also shows that topologies with higher degrees tend to have lower aggregation rates. This is because in a network with denser connections, backup paths tend to be more scattered and the possibility to aggregate them decreases.

### C. Scalability

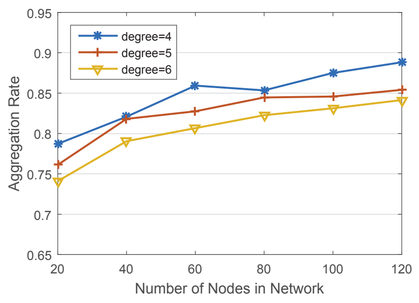
We now evaluate the scalability of our scheme by carefully examining the computing time of each module, including the time for the backup path construction (IBFS) and for the backup path aggregation (2SA). To this end, we first apply the scheme to the example topologies with increasing scales in



(a) Backup rule numbers on example topologies.



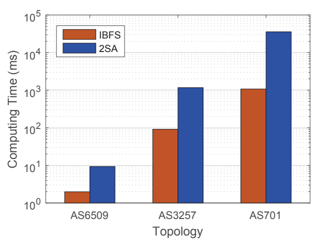
(b) Backup rule number on each switch.



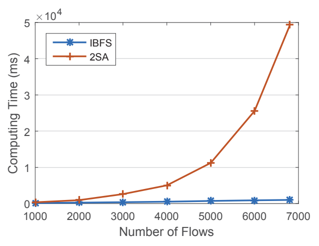
(c) Aggregation rates (BRITE).

Fig. 6. Results on real-world topologies and 20-120-node generated topologies.

Table III for the maximum number of source-destination flows, then vary the number of flows in the largest topology, AS701, to observe the changes of the computational time. The results, which are the average values of 10 random experiments, are shown in Fig. 7.



(a) Computing time for example topologies.



(b) Computing time for varying number of flows.

Fig. 7. Time of backup path construction (IBFS) and aggregation (2SA).

In Fig. 7(a), we can see that the computing time of both IBFS and 2SA increases rapidly as the topology scales up. This is because, in larger topologies, more backup paths need to be constructed and aggregated. However, even in the largest topology, AS701, both IBFS and 2SA can always terminate within 60 seconds. And our scheme can usually finish in several microseconds for small topologies. On the other hand, even in a large topology, when the number of flows remains relatively small, our scheme can also perform well with a low latency, as shown in Fig. 7(b). Besides, both the figures reveal that the main contributing factor to the latency overhead in our scheme is the backup path aggregation module. This result matches the properties of the IBFS and 2SA algorithms. We consider that the longer latency imposed by the 2SA algorithm is acceptable since we do not need to run it in real time. To conclude, with a reasonable latency overhead, our scheme is applicable to large-scale networks.

## VII. Conclusion

In this paper, we propose a backup scheme to recover from any single-link failure in SDN networks. The scheme can provide customized backup paths. Meanwhile, it achieves fast recovery and low storage overhead on flow tables. Our evaluation shows that the proposed scheme reduces 63.33%-97.55% backup rules compared with the traditional flow-based protection. It also shows that our scheme has high availability.

## Acknowledgment

The research is supported by the National Natural Science Foundation of China under Grant 61625203 and the National Key R&D Program of China under Grant 2016YFC0901605, the R&D Program of Shenzhen under Grant Shenfagai(2015)986.

## References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM CCR*, vol. 38, no. 2, pp. 69–74, 2008.

[2] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM CCR*, vol. 43, no. 4, pp. 3–14, 2013.

[3] Q. Yan and F. R. Yu, "Distributed denial of service attacks in software-defined networking with cloud computing," *IEEE Communications Magazine*, vol. 53, no. 4, pp. 52–59, 2015.

[4] J.-P. Vasseur, M. Pickavet, and P. Demeester, *Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS*. Elsevier, 2004.

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

[6] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, "Openflow-based segment protection in ethernet networks," *Journal of Optical Communications and Networking*, vol. 5, no. 9, pp. 1066–1075, 2013.

[7] N. L. M. v. Adrichem, B. J. v. Asten, and F. A. Kuipers, "Fast recovery in software-defined networks," in *2014 Third European Workshop on Software Defined Networks (EWSDN)*, 2014.

[8] "Rocketfuel: An isp topology mapping engine." [Online]. Available: http://research.cs.washington.edu/networking/rocketfuel

[9] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: An approach to universal topology generation," in *2001 9th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOT)*, 2001.

[10] M. Hou, D. Wang, M. Xu, and J. Yang, "Selective protection: A cost-efficient backup scheme for link state routing," in *2009 29th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2009.

[11] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[12] "Ryubook 1.0 documentation." [Online]. Available: http://osrg.github.io/ryu-book/en/html