THRIFTY: Towards High Reduction In Flow Table memorY

Ali Malik
School of Computing, University of Portsmouth, United Kingdom
ali.al-bdairi@port.ac.uk

Benjamin Aziz
School of Computing, University of Portsmouth, United Kingdom
benjamin.aziz@port.ac.uk

Chih-Heng Ke
Department of Computer Science and Information Engineering, National Quemoy University, Taiwan
smallko@nqu.edu.tw

Abstract

The rapid evolution of information technology has compelled the ubiquitous systems and computing to adapt with this expeditious development. Because of its rigidity, computer networks failed to meet that evolvement for decades, however, the recently emerged paradigm of software-defined networks gives a glimpse of hope for a new networking architecture that provides more flexibility and adaptability. Fault tolerance is considered one of the key concerns with respect to the software-defined networks dependability. In this paper, we propose a new architecture, named THRIFTY, to ease the recovery process when failure occurs and save the storage space of forwarding elements, which is therefore aims to enhance the fault tolerance of software-defined networks. Unlike the prevailing concept of fault management, THRIFTY uses the Edge-Core technique to forward the incoming packets. THRIFTY is tailored to fit the only centrally controlled systems such as the new architecture of software-defined networks that interestingly maintain a global view of the entire network. The architecture of THRIFTY is illustrated and experimental study is reported showing the performance of the proposed method. Further directions are suggested in the context of scalability towards achieving further advances in this research area.

2012 ACM Subject Classification Networks → Network protocols

Keywords and phrases Source Routing, Resiliency, Fault Tolerance, SDN, TCAM

Digital Object Identifier 10.4230/OASIcs.ICCSW.2018.2

Category Main Track

1 Introduction

Computer networks play an essential role in changing the life style of modern society. Nowadays, most of the Internet services are located in data centers, which are consisting of thousands of computers that connected via large-scale data center networks. Typically, wide-area networks interconnecting the data centers that distributed across the globe. The Internet users are usually using their devices (i.e. computer, mobile, tablet, smart watch, etc.) to access the various available services of Internet through different ways such as WiFi, Ethernet and cellular networks. Traditionally, the distributed control systems in networking devices along with a set of defined protocols (e.g. OSPF [16] and RIP [9]) constitute a fundamental technology that have been adopted to send and receive data via networks

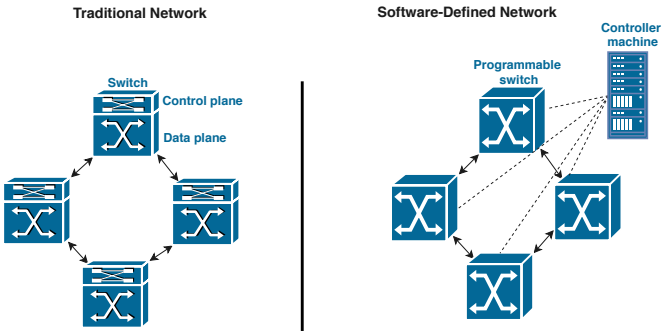


Figure 1 Traditional versus SDN architecture.

around the world in recent years. According to [2], these distributed protocols increase the inflexibility of network management through making the network operators to lose their visibility over their networks. Managing the networks efficiently to meet the requirements of the Quality of Service (QoS) and the Service Level Agreements (SLA) are the core challenging points of the computer networks, which need to be improved continuously in light of the increasing number of devices that are connected to the Internet, which are currently estimated to be in the range of 9 billion devices and expected to reach double that number by 2020. Therefore, the Internet ossification is highly expected as stated in [12]. One possible solution is to replace the complex/rigid networking system with an open and programmable network instead. Software-Defined Networking (SDN) is a promising paradigm that resulted from a long history of efforts aiming to simplify the computer networks management and control [5]. In SDN the *control plane* has been decoupled from the *data plane* and placed in a central location usually called the network *controller* or the network operating system. Figure 1 illustrates the difference between the SDNs and conventional networks architecture. Such a new networking architecture of SDN with much more flexibility comparing to the traditional networks meant that SDNs are nowadays adopted by many of the well known pioneering companies like Deutsche Telekom, Google, Microsoft, Verizon, and CISCO, which have recently combined in 2011 to launch the Open Network Foundation (ONF) [18] as a nonprofit consortium that aims to accelerate the adoption of SDN technologies.

Although SDNs have brought many advantages with dramatic network improvements, this innovation has been accompanied by several challenges, such as the management of network failures and updating of the network architecture [1].

2 Related Work

Since link and node failure is an issue almost as old as computer networks, so far, SDN follows the traditional fundamental strategies of failure recovery (i.e. protection and restoration) to recover from the data plane failures. However, the fault management in SDNs differs from the legacy networks in the way of computing and update the routing tables. Instead of the conventional way of reconfiguration in which each node makes the required changes to update the routing table locally, the controller in SDN is responsible to handle the network reconfiguration and instruct the relevant nodes on how to follow the new update, which is therefore made globally. Protection and restoration are currently the only two ways to reconfigure the network and mask failure incidents. However, each associated with some drawbacks in terms of time and memory space consumption. In protection, the alternative solutions (i.e. backups) are preplanned and installed in the relevant switches, however, in

restoration the possible solutions are not preplanned and will be calculated dynamically when failure occurs. A large number of studies have considered the issue of network failures and propose different contributions that are reviewed in [6]. Unfortunately, the current SDN switches in the market have a limited capacity of flow table due to the small space of the expensive Ternary Content-Addressable Memory (TCAM) [10]. Recently, this issue took place in the proposed schemes of failure recovery as the new schemes should consider the problem of TCAM limitation.

In this context, [19] propose SlickFlow, a source-based routing method to enhance the scalability and fault tolerance in OpenFlow networks. In SlickFlow, the controller computes the primary and the backup (disjoint) paths and then both are encoded in the packet header in addition to an *alternative* bit, which indicates the current using path. When the primary path is affected by link failure then, a switch will forward the packets through the backup path and change the value of the alternative, which is necessary for the neighbor switch to follow the backup as well. The packet header provides an additional limited segment of information that can be used for the purpose of encoding path details [17], where the alternative path should not exceed 16 hops.

The authors in [15] produce a protection scheme, as an extension to their previous work in [14], that minimises TCAM consumption. The authors developed two routing strategies: *Backward Local Rerouting* (BLR) and *Forward Local Rerouting* (FLR). In BLR, a node-disjoint path is computed as a backup for every primary path in the network and when a failure occurs, packets are sent back to the origin node to be rerouted over the pre-planned backup towards the destination. In FLR, a backup route for each link in the primary path is pre-computed. When a link failure occurs, the packets will be forwarded from the point of failure towards the downstream switch in the primary path by following the backup path, however, in case of there will be a multiple backups then, the one with least number of switches will be chosen. Instead of using fast failover group type, the authors have extended the OpenFlow protocol by adding an additional entry that called `BACKUP_OUTPUT` to the `ACTION SET` of the flow table entries, so that the new added entry is responsible to set the out put port when a link fails.

The authors in [23] propose a new flow tables compression algorithm as well as a compression-aware routing concept to enhance the ratio of the gained compression rate. The proposed algorithm reduces the consumed TCAM space by using the wildcard to match the tables who shared the same output and packet modification operations and hence the compression. The authors relied on their previous work [22] in which they proposed Plinko as a new forwarding model where the forwarding table entries apply the same action.

The authors in [24] discuss the problem of the protection schemes and its impact on the shortage of TCAM memory. The authors proposed *Flow Entry Sharing Protection* (FESP), which is a greedy algorithm that selects the node with larger available flow entry capacity and minimum backup flow entry. The study showed how the total number of flow entries can be minimised where the experimental results revealed that the reduction ratio of flow entries is up to 28.31% compared with the existing path and segment protection methods. With respect to all contributions, some issues still exist such as the following:

1) The disjointness as constraint for the calculated backups will require a totally new set of flow entries, which in turn will consume an additional TCAM space.

2) Compress flow tables using wildcard will affect the fine-grained per packet inspection and therefore might lead to policy/security violations.

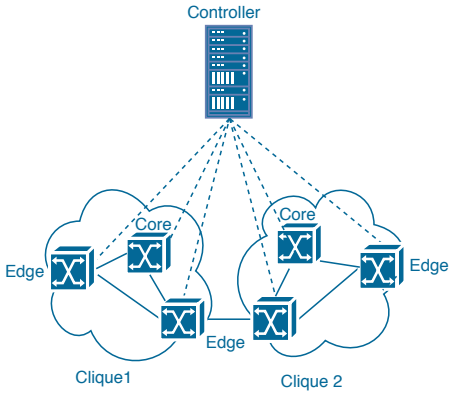


Figure 2 THRIFTY architecture.

3 Problem Statement

On one hand, protection solutions require an additional information, which have to be loaded into the data plane elements, to tell the nodes how to perform when failure occurs. However, the extra loaded information affects the storage memory of the network switches and therefore the designed fault tolerance mechanisms should consider the limited space of flow table and TCAM. On the other hand, it is very hard to meet the carrier-grade reliability requirements (i.e. recover within 50 μs) in restoration [20, 21] because the infrastructure layer equipment in SDN are dummy forwarding elements due to the split architecture, then, the central controller is responsible for calculating the alternative paths and then installing the flow entries (i.e. forwarding rules) in the relevant switches of each backup after detecting failures.

4 THRIFTY for SDNs

THRIFTY is a scalable fault tolerant system with aim to reduce the TCAM storage space of forwarding elements as much as possible. THRIFTY has the following properties:

- *Edge-Core based routing*: The idea of Edge-core design has been proposed in [4], in which the complex control functions have been removed to the ingress switches and keep the remain core switches as clean-slate. THRIFTY makes use the same idea of Edge-core design and to be applied on the partitioned network topology, as an extension to our previous work in [13] in which the network topology can be divided into N number of cliques.
- *Fast recovery*: Reacting to network link failures, THRIFTY is capable to recover from single/multi link failures in a carrier-grade time scale (i.e. less than 50 μs).
- *Scalable to large-scale networks*: As the size of network topology increases, the flow table entries of data plane will be still manageable due to the designed architecture.
- *Single network controller*: Network can be controlled by one controller and it is the entity that responsible for the network activities and adjust the global policy of network.

4.1 Architecture

Figure 2 depicts THRIFTY architecture, the controller comprises three modules, each responsible for a specific task as follows:

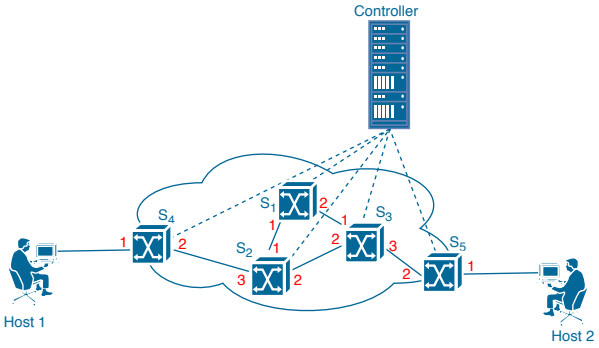


Figure 3 Example topology.

1) *Topology parser*: is responsible for fetching the underlying network topology characteristics and build a topological view in order to represent the gained network topology as a graph G, we utilised the NetworkX [8] tool, which is a pure python package with a powerful set of functions that can be used to manipulate and simplify network graphs.
2) *Cliques producer*: is responsible for partitioning the constructed network graph G into set of sub-graphs by incorporating the well known community detection algorithm *Girvan and Newman* [7] to produce a set of possible cliques (with any size). The densely connection between the resulted cliques' vertices is the main interesting feature of Girvan and Newman algorithm, in other words, the strong connection among the nodes in each clique could provide a multiple alternative paths that could be utilised when failures occur.
3) *Edge-Core finder*: Based on the resulted cliques, this module is responsible for dividing the set of nodes, in each single clique, into two sets *Edge* and *Core*. Therefore, we will have two kind of switches, namely Edge and Core. The key challenging point of this module is to find the optimal number of Edge switches.

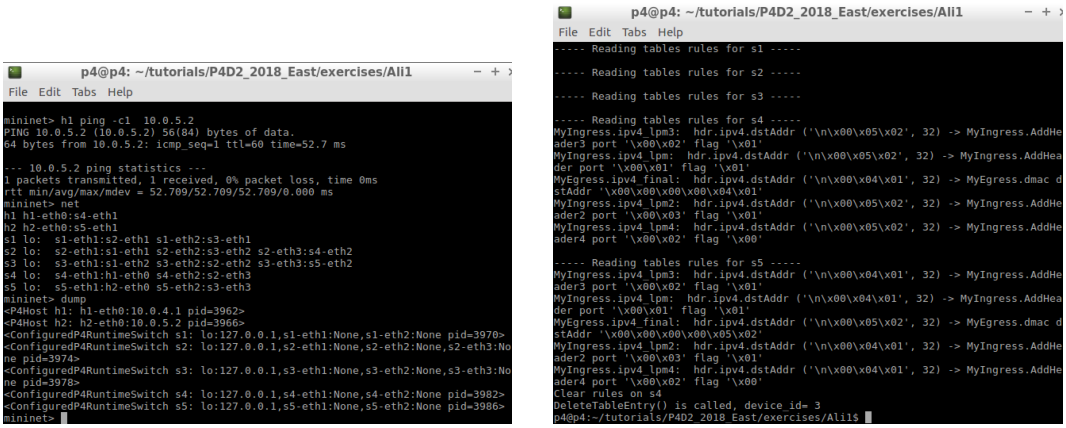
4.2 Prototype and Implementation

To demonstrate the feasibility of the proposed architecture, we provide a prototype implementation of THRIFTY. The current prototype is designed as a proof of concept as well as to show how the proposed solution can be applied.

The current implementation of THRIFTY is prototyped with the recently proposed P4 language [3] using a software switch as a platform. We use the open source P4[1] as a packet processing language to create a set of P4 switches in the specified topology of Figure 3 in which *Edge*={`S4, S5`} and *Core*={`S1, S2, S3`}. We evaluate our THRIFTY prototype using Mininet [11] as a virtual network emulator, which is suitable to generate customized virtual network topologies in a single Linux machine. The current implementation is divided into two schemes as follows:

1. Rules aggregation method ($Scheme_1$)[2]
   In this method, the necessary flow entries (from source to destination) of a particular path are stored in Edge switches of the network in addition to add one more flow entry

[1] `P4 switch model available at: https://github.com/p4lang`
[2] `The implementation can be found at: http://csie.nqu.edu.tw/smallko/sdn/mysource_routing.htm`



(a) Ping test. **(b)** Required rules.

Figure 4 Adding rules with Scheme$_1$.

in each Edge switch for the purpose of changing the destination mac address. Therefore, the number of required flow entries in this method can be calculated by:

`The number of traversed switches in a path + 1`

Figure 4 shows the preliminary results of this method. Although the scheme fails to reduce the number of required rules, it is still of interest since it collects the required rules in two locations (i.e. Edges) rather than distributed them over switches and therefore it might increase the flexibility of updating the network.

2. Rules compression method ($Scheme_2$)[3]
   In this method, the entire flow entries from source to destination of a particular path are reduced to one rule only, which also need to be stored in one of the path Edge switches. For instance, in the given example topology the shortest path between `Host1` and `Host2` is:

   `Host1-S4-S2-S3-S5-Host2`

   we set the next couple of rules to indicate the routing information at the ingress switch (`S4`):

   ```
   table_add ipv4_lpm set_path 10.0.5.2/32 => 4 1 3 2 2 0 0 0 0 0
     table_add ipv4_final dmac 10.0.4.1/32 => 00:00:00:00:04:01
   ```

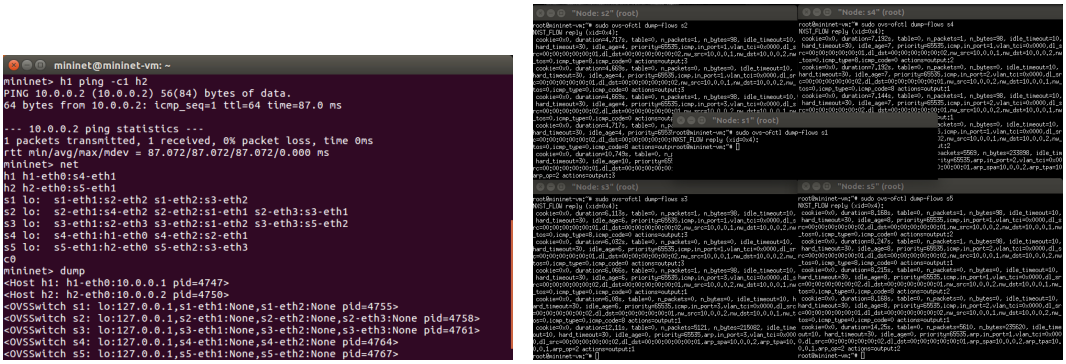
   While, in the egress switch (`S5`) we had the following;

   ```
   table_add ipv4_lpm set_path 10.0.4.1/32 => 4 1 3 2 2 0 0 0 0 0
     table_add ipv4_final dmac 10.0.5.2/32 => 00:00:00:00:05:02
   ```

   Where 4 indicates that the shortest path between `Host1` and `Host2` contains 4 nodes (i.e. 3 hops). While, the rest of digits denotes the set of output ports for the switches along the path as follow: 1 refers to the output port of `S5`, 3 refers to the output port of `S3`, 2 refers to the output port of `S2` and the last 2 refers to the output port of `S4`.

In order to compare our proposed methods to traditional/existing method, we replicated the same experimental procedure with POX controller where the *openflow.discovery*[4] and

[3] `The implementation can be found at:`
`http://csie.nqu.edu.tw/smallko/sdn/mysource_routing2.htm`
[4] `https://github.com/att/pox/blob/master/pox/openflow/discovery.py`



(a) Ping test. **(b)** Required rules.

Figure 5 Adding rules with traditional scheme.

l2_multi[5] modules have been utilised to discover and setup the shortest path from sender to receiver. Figure 5 shows the number of rules required to forward the incoming packets from `Host1` to `Host2`. It is worth mentioning here that we took into account the only IP packets, however, the arp packets have been discarded.

As a result, Figure 6 illustrates the total number of flow rules required by the three simulated schemes (i.e. $Scheme_1$, $Scheme_2$ and traditional).

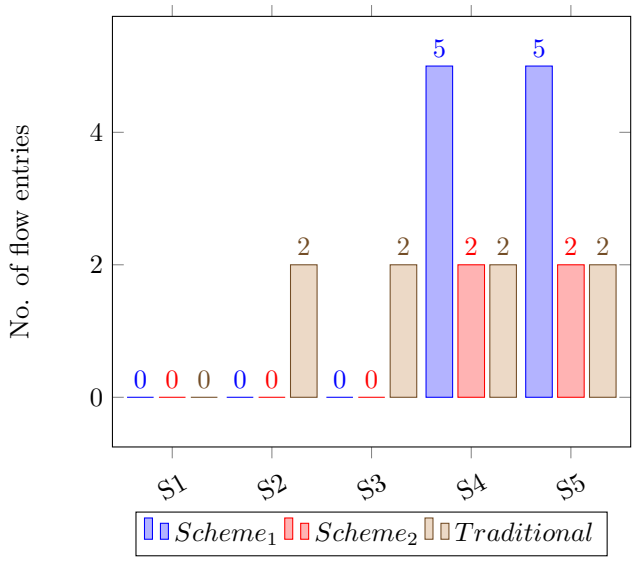


Figure 6 A comparison of three scenarios.

5 Conclusion and Future Work

In this paper, we presented the ongoing work in realising THRIFTY as a new solution to tackle the TCAM limitation problem as well as to accelerate the recovery from link failures. We showed how the proposed solution can be implemented using a couple of new schemes

[5] `https://github.com/att/pox/blob/master/pox/forwarding/l2_multi.py`

that aggregate and compress the forwarding rules. As a future plan, the authors will proceed to conduct failure scenarios in addition to extend the current piece of work by considering the following aspects:

Building a general framework: THRIFTY uses Edge-Core architecture on the basis of cliques concept with a view of dramatically simplifying the packet forwarding as well as reducing the number of flow table entries that will enhance the SDN scalability.

Dependability attributes: Currently, THRIFTY only supports the scenario of data plane link failures, however, our work envisions to include other attributes of dependability such as security.

References

1 Ian F Akyildiz, Ahyoung Lee, Pu Wang, Min Luo, and Wu Chou. A roadmap for traffic engineering in SDN-OpenFlow networks. *Computer Networks*, 71:1–30, 2014.

2 Theophilus Benson, Aditya Akella, and David A Maltz. Unraveling the Complexity of Network Management. In *NSDI*, pages 335–348, 2009.

3 Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

4 Martin Casado, Teemu Koponen, Scott Shenker, and Amin Tootoonchian. Fabric: a retrospective on evolving SDN. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 85–90. ACM, 2012.

5 Nick Feamster, Jennifer Rexford, and Ellen Zegura. The road to SDN: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, 44(2):87–98, 2014.

6 Paulo Fonseca and Edjard Mota. A survey on fault management in software-defined networks. *IEEE Communications Surveys & Tutorials*, 2017.

7 Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

8 Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using NetworkX. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

9 Charles L Hedrick. Routing information protocol. Technical report, Rutgers University, 1988. https://tools.ietf.org/html/rfc1058.

10 Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

11 Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19. ACM, 2010.

12 Pingping Lin, Jun Bi, Hongyu Hu, Tao Feng, and Xiaoke Jiang. A quick survey on selected approaches for preparing programmable networks. In *Proceedings of the 7th Asian Internet Engineering Conference*, pages 160–163. ACM, 2011.

13 Ali Malik, Benjamin Aziz, Chih-Heng Ke, Han Liu, and Mo Adda. Virtual topology partitioning towards an efficient failure recovery of software defined networks. In *The 16th International Conference on Machine Learning and Cybernetics (ICMLC)*. IEEE, 2017.

14 Purnima Murali Mohan, Tram Truong-Huu, and Mohan Gurusamy. TCAM-aware local rerouting for fast and efficient failure recovery in software defined networks. In *Global Communications Conference (GLOBECOM), 2015 IEEE*, pages 1–6. IEEE, 2015.

15 Purnima Murali Mohan, Tram Truong-Huu, and Mohan Gurusamy. Fault tolerance in TCAM-limited software defined networks. *Computer Networks*, 116:47–62, 2017.

16 John Moy. OSPF version 2. Technical report, Ascend Communications, Inc., 1998. https://tools.ietf.org/html/rfc2328.

17 Giang TK Nguyen, Rachit Agarwal, Junda Liu, Matthew Caesar, P Godfrey, and Scott Shenker. Slick packets. *ACM SIGMETRICS Performance Evaluation Review*, 39(1):205–216, 2011.

18 ONF. Open Networking Foundation, 2018. https://www.opennetworking.org/.

19 Ramon Marques Ramos, Magnos Martinello, and Christian Esteve Rothenberg. Slickflow: Resilient source routing in data center networks unlocked by openflow. In *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, pages 606–613. IEEE, 2013.

20 Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet, and Piet Demeester. Enabling fast failure recovery in OpenFlow networks. In *Design of Reliable Communication Networks (DRCN), 2011 8th International Workshop on the*, pages 164–171. IEEE, 2011.

21 Dimitri Staessens, Sachin Sharma, Didier Colle, Mario Pickavet, and Piet Demeester. Software defined networking: Meeting carrier grade requirements. In *Local & Metropolitan Area Networks (LANMAN), 2011 18th IEEE Workshop on*, pages 1–6. IEEE, 2011.

22 Brent Stephens, Alan L Cox, and Scott Rixner. Plinko: Building provably resilient forwarding tables. In *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, page 26. ACM, 2013.

23 Brent Stephens, Alan L Cox, and Scott Rixner. Scalable multi-failure fast failover via forwarding table compression. In *Proceedings of the Symposium on SDN Research*, page 9. ACM, 2016.

24 Xiaoning Zhang, Shui Yu, Zhichao Xu, Yichao Li, Zijing Cheng, and Wanlei Zhou. Flow Entry Sharing in Protection Design for Software Defined Networks. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–7. IEEE, 2017.