When Software Defined Networks Meet Fault Tolerance: A Survey

Jue Chen, Jinbang Chen(✉), Fei Xu(✉), Min Yin, and Wei Zhang

School of Computer Science and Software Engineering,
East China Normal University, Shanghai 200062, China
jue0428@126.com, {jbchen,fxu}@cs.ecnu.edu.cn

Abstract. Software Defined Network (SDN) is emerging as a novel network architecture which decouples the control plane from the data plane. However, SDN is unable to survive when facing failure, in particular in large scale data-center networks. Due to the programmability of SDN, mechanism could be designed to achieve fault tolerance. In this survey, we broadly discuss the fault tolerance issue and systematically review the existing methods proposed so far for SDN. Our representation starts from the significant components that OpenFlow and SDN brings – which are useful for the purpose of failure recovery, and is then further expanded to the discussion of fault tolerance in data plane and control plane, in which two phases – detection and recovery – are both needed. In particular, as the important part of this paper, we have highlighted the comparison between two main methods – restoration and protection – for failure recovery. Moreover, future research issues are discussed as well.

Keywords: Software defined network · Fault tolerance · OpenFlow · Failure detection · Recovery · Restoration · Protection

1 Introduction

When the Internet was stretched even though continuous efforts were made to upgrade devices or protocols, an innovative network architecture – Software Defined Network (SDN), has gained much attention in both academia and industry. With the ability to decouple the data plane from the control plane, SDN brings much benefit to commercial enterprises [13,20]. For instance, Google's B4 – a globally-deployed software defined WAN [15], has successfully improved the utilization of network links up to nearly 100 percent, which is threefold as before. As for any network systems, including Google's geo-distributed data-centers B4, network faults are prevalent, which can inevitably cause catastrophic effects to user applications. Therefore, to consider and further to solve every possible malfunction becomes a need for all network systems. Accordingly, one of the urgent needs for SDN is to provide the ability of failure recovery. For this issue, we focus ourselves on the breakdown of network components in this article,

G. Wang et al. (Eds.): ICA3PP 2015, Part III, LNCS 9530, pp. 351–368, 2015.
DOI: 10.1007/978-3-319-27137-8_27

while excluding the discussion of network update [44] such as forwarding loop and forwarding black hole.

In traditional networks, such as the Internet, routing protocols can help convergence of the whole network automatically. However, as the "brain" of SDN, a controller does not have the ability of self-healing when initially designed and introduced, and it must be equipped with the capacity of fault tolerance. In SDN, the fault tolerance issue covers across two different levels – the data plane and the control plane.

On one hand, if a link breaks or a switch meets an outage in the forwarding plane, a controller needs to help to find another valid routing path to continuously deliver packets as there is almost no intelligence on network devices – which is called the failure recovery of data plane. Chronologically, this procedure consists of two phases – failure detection and failure recovery. Furthermore, two typical methods – restoration and protection [43] are commonly used in the recovery process. With restoration, switches must alert the controller to the fault, then the controller will generate certain commands to direct data plane to update their forwarding tables. On the contrary, with protection, switches would establish table entries for two paths – the working path and the backup path before the fault occurs. If the switch detects the malfunction, it can switch over to the protection path automatically without the participation of the controller. At the moment, most papers in this area are focusing on solving a single link failure. As a result, the solutions for multiple link failures and the outage of a switch remain a challenge.

On the other hand, the fault tolerance of control plane is of much concern as well. Controllers need to take charge of the whole network all the time. As a result, the diagram of multiple controllers have been proposed to ensure the reliability of the control plane [25]. In addition, the OpenFlow channel between controllers and switches may also meet fault. If a switch loses connection with the controller, another path is needed which may walk through other neighbours to reach the controller [38]. Nearly no papers are focusing on solving the failure of the OpenFlow channel and it could be regarded as a future research direction. In summary, both devices and links need to be provided with redundancy to cope with a variety of failures in the context of SDN.

The rest of this survey is organized as follows. Section 2 presents some preparatory work for fault tolerance in SDN, which is based on the main idea of evolution from the traditional networks to SDN, including existing strategies (mostly heuristic) of failure recovery in traditional networks, and essential structures in OpenFlow [27] protocol which are used for fault tolerance of SDN. The existing methods as well as comparison, summaries and future research issues are then discussed in the following three sections. We review the failure detection of data plane in Sect. 3, and then discuss the failure recovery of data plane and control plane in Sects. 4 and 5 respectively. Finally, a conclusion is given in Sect. 6.

2 Prerequisites for Fault Tolerance in SDN

There are preparatory works related to the fault tolerance in SDN. In this section, we first summarize the experience of failure recovery in traditional networks, and

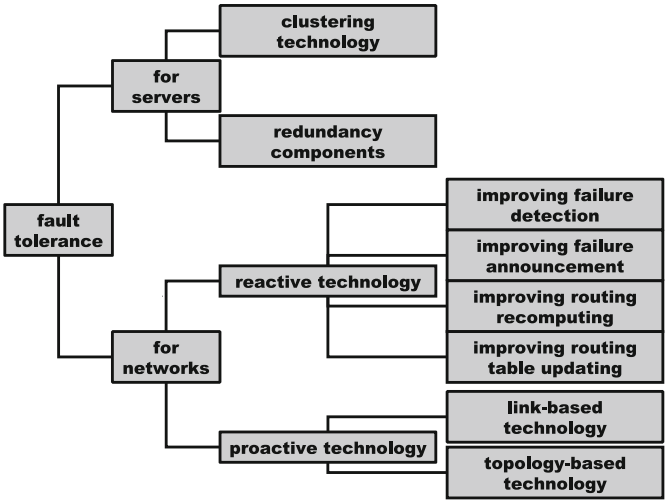

Fig. 1. Classification of fault tolerance methods in traditional networks

then provide the concepts of flow table and group table which are the bases of the OpenFlow protocol.

2.1 A Review of Fault Tolerance in Traditional Networks

In general, each network component may face a failure. Here we mainly discuss fault tolerance on servers and links for traditional networks [35], which are summarized and classified in Fig. 1.

Based on the difference with respect to the number of nodes (servers), the fault tolerance on servers can be separated into two parts. When using the clustering technology, multiple computers cooperate with each other. As long as at least one of them works correctly, the whole system can provide services to users. On the other hand, in the situation that only one server is in use, equipping each component (including CPU, memory, disk, network card and even power supply, *etc.*) with a backup is effective.

Traditional routing protocols (such as OSPF [28]) may need tens of seconds to converge. However, in the carrier grade networks, the recovery time is required to be within millseconds. Considering the fault tolerance which consists of four steps – failure detection, failure announcement, route recomputing and routing table updating, efforts have been made to decrease the running time of each procedure to reduce the whole recovery time. Firstly, methods such as reducing the transmit interval of probing packets [8] or combining with other detection methods (Bidirectional Forwarding Detection [17]) can be used to cut down the failure detection time. Secondly, when considering the transient failure, a self-adapting timer [9], for example, can be adopted to avoid announcement of this fault. Thirdly, routing algorithms can be improved. Finally, the batch update,

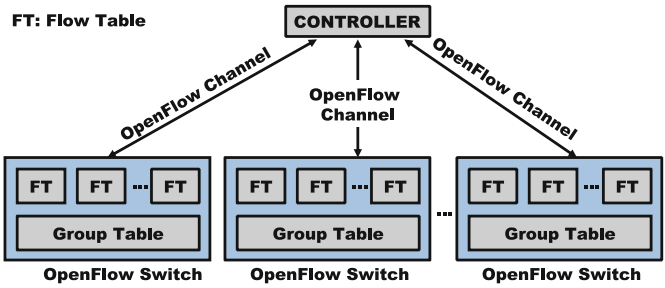

Fig. 2. Architecture of OpenFlow protocol

for instance, can accelerate the update of forwarding tables. This technology is called the reactive failure recovery.

The reactive failure recovery is initialized after failures happen. In order to further decrease the recovery time, the proactive technology is proposed. With this method, resources have been preserved and backup paths have been worked out [21] before the failure happens, leading to the benefit that protection paths could come into use immediately once the faults are detected. According to the difference in the scale of malfunction, the proactive mechanism can be classified into link-based and topology-based technology, aiming at a single failure or multiple failures respectively. The former mainly focuses on protecting certain links or devices, such as Loop-Free Alternates (LFA) algorithm [7] and Not-Via algorithm [11]; while the latter will calculate a few logical backup topologies which can help to protect the whole network [31,32,45]. In comparison, the proactive technology performs better on recovery time as a few preparatory works have been done before failures happen. However, these works will spend extra network resources which can not be neglected especially in the overloaded networks.

2.2 What Does OpenFlow Bring for Fault Tolerance

The architecture of SDN can be divided into two parts: the control plane and data plane. The control plane is composed of the controller and the channels between the controller and network elements; while the data plane contains switches and hosts [18]. Besides, the OpenFlow protocol (Fig. 2 shows the architecture of this protocol) is widely used for communicating between the two planes, and most of the methods mentioned in this paper are based on this *de facto* southbound protocol. In SDN, switches can only forward packets according to the forwarding rules established and updated by the controller.

This paper mainly focuses on how to use the OpenFlow protocol to solve the fault tolerance problem in SDN. At first a few concepts of this new protocol are introduced, including the flow table and group table. Each flow table contains a set of table entries, and each table entry consists of match fields, counters, and a set of instructions which are deployed for packets matching [4]. Firstly, each match field is composed of tuples, consisting of ingress port, source/destination

MAC address, source/destination IP address, port numbers, *etc.* Moreover, a flow can be defined according to arbitrary combinations of these elements, unlike the traditional routing protocols which are based on the destination IP address only. Secondly, the counters are used to count the number of packets/bytes of a flow. Finally, the instructions describe how to process these packets such as forwarding, dropping, redirecting to the group table and so on.

When a flow table entry points to a group, the concept of group table needs to be described. A group table consists of several group table entries, and each entry contains: group identifier, group type, counters and action buckets. Firstly, the group identifier uniquely identifies each group. Secondly, the group type determines the group semantics, including the following types: "all", "select", "indirect" and "fast fail over". Thirdly, the counters are used to count the number of packets/bytes. Finally, the action buckets contain a set of actions and associated parameters. The principle of the "fast fail over" group type is to execute the first live bucket every time which can be applied in the fault tolerance. Multiple paths of an identical flow can be preserved into the same group table entry with the working path in the first bucket. When the working path fails, the following buckets of protection paths (if alive) can be used, enabling the switch to automatically change forwarding route without informing the controller [5].

3 Fault Tolerance on Data Plane: Phase 1-Detecting Link Failures

To guarantee the availability of the data plane, the source host is required to communicate with the destination even though link failures happen. When referring to the fault tolerance on this plane, the two steps – detecting and recovering from failures, are needed which will be described in the following two sections, respectively.

3.1 Methods

Loss Of Signal (LOS, as shown in Fig. 3(a)), which is widely used to detect link failures in carrier-grade networks [39], can be utilized by switches to perceive the status change of each port. In SDN, this warning message need to be transferred to the controller for further processing. Similarly, the method proposed in [22] (as shown in Fig. 3(c)) is also applied to detect link failures. The main idea of this approach is using a circle which starts and terminates at the controller to monitor the status of a few links. Under normal circumstances, the control packet is transmitted along the loop, and finally return to the controller. However, if there is a failure inside the circle (which means a link breaks), the second stage initiates where each switch in the loop is required not only to deliver the packet to the next hop, but also send back to the controller. As the result, the link failure can be located.

Sometimes detecting path failures is needed where a path is composed of multiple links. Bidirectional Forwarding Detection (BFD) [17] (as shown in Fig. 3(b))

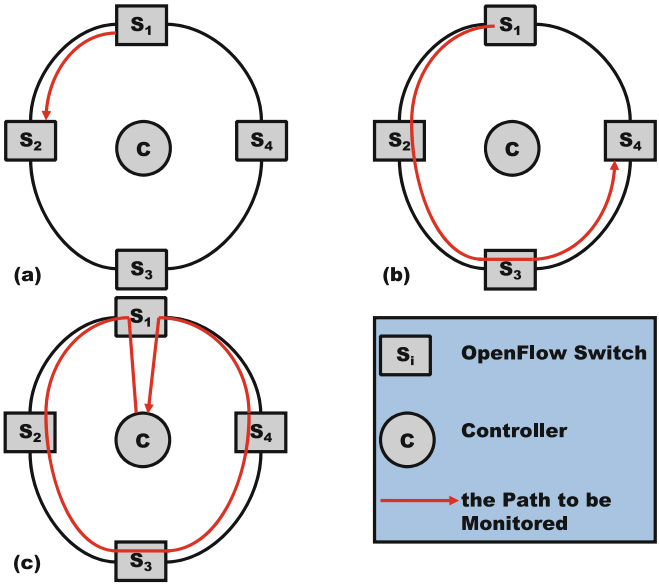

Fig. 3. Detection methods (a) LOS through monitoring a link (b) BFD through monitoring a path (c) two methods through monitoring a circle

is designed for this purpose. It is a Hello protocol and the two end nodes of a BFD session transmit echo packets periodically. If a system stops receiving the packet from the monitored connection, the path may be assumed to be broken [39].

With the number of devices increases, multiple controllers are needed to cooperate to control the whole network. In this situation, each controller has its own control domain. However, considering packets can pass through arbitrary links, the controller may need to be informed of a failure outside of its region. [19] (as shown in Fig. 3(c)) suggests using a circle to help locating link failures which is similar to the approach proposed in [22], but the difference is when a fault is detected, the binary search is adopted so that the hunting zone decreases until a single broken link is figured out. Moreover, the two methods by using a circle need extra flow table entries to return packets to the controller, or change the walking direction.

3.2 Comparison and Summaries

A comparison is summarized in Table 1 with some extended discussion. As the methods mentioned before are executed in different experimental environment, the detection time doesn't have comparability, but the factors affecting the detection time can be discussed. It can be concluded that each method is related to the length of the monitoring path, and the LOS is relatively stable as its granularity is a link.

Table 1. Comparison of failure detection methods

Approaches	LOS [40]	BFD [39]	[22]	[19]
Factors affecting detection time	×	Length of path	Length of circle	Length of circle
Location of a single link failure	✓	×	✓	✓
Location of multiple link failures	✓	×	×	×
Number of failure detection messages	0	1 packet from each switch in each time interval	The number of rounds	$1 + \lceil log_2(L_{opt}) \rceil$
Using in-band connection	×	✓	✓	✓
Requirement of extra flow table entries	×	×	✓	✓
Applying to multiple controllers	×	×	×	✓

Location of failures means finding out the specific broken link or links, more than just detecting failures in the data plane. Firstly, BFD can only judge whether the whole path is normal or not. Secondly, the two methods by the use of a circle can only locate the first failed link near the controller. Finally, by using LOS, each switch will generate port status message to the controller independently when its link has broken, then each link can be located separately. As the result, only the LOS can locate multiple link failures.

Summaries. When choosing an appropriate failure detection method, the recovery technology (restoration or protection) to be used in combination can be considered. If the restoration mechanism is adopted which means informing the controller after the link breaks, the LOS which will generate port status message to the controller is a good choice. If the protection is in use, the BFD can be applied to protect each path without the involvement of the controller. Specifically in the situation where a number of hosts want to communicate with each other, multiple paths must be established and observed for all the pairs when adopting BFD. In order to reduce the number of BFD sessions as well as the detection time, establishing BFD sessions for per link can be a choice as the number of links is relatively less and more stable compared to the number of paths when BFD sessions are set up for each path.

4 Fault Tolerance on Data Plane: Phase 2-Recovering from Link Failures

The recovery methods are divided into restoration and protection, respectively. In this section, these two technologies are introduced firstly, and next open research issues will be discussed.

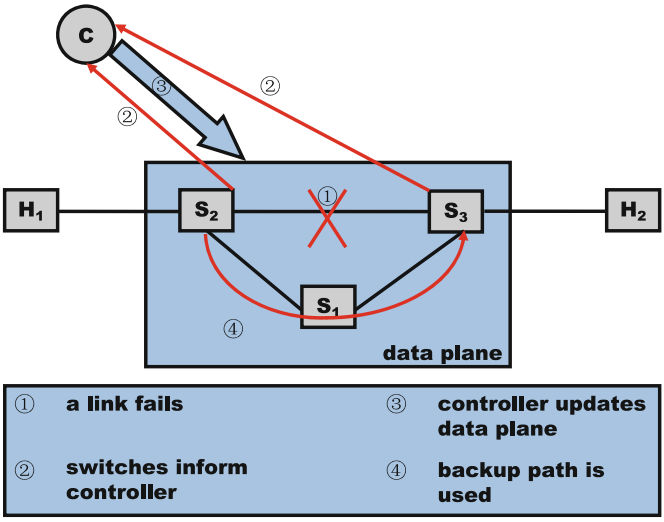

Fig. 4. Restoration technology

4.1 Restoration

Considering the situation where a working path is broken, the restoration technology is a choice which can be used to recover from link failures. Generally speaking, This technology can be divided into three steps (shown in Fig. 4): 1. The relevant switches inform the controller about the change of a port's status. 2. The controller can either work out another path after the link failure [40], or has calculated the protection path along with the working path before the failure. In the latter situation, the protection path can be saved in switches' flow tables [29], or stored inside the controller [24]. Regardless of the methods adopted, the controller must send packets out to update switches' flow tables. 3. Data flows will go through the protection path.

In detail, the method proposed in [40] uses the shortest path algorithm to calculate route before the link failure. It can be integrated with the LOS. When the "port down" event arrived at the controller, the controller can find the flows affected by the failed link, and recompute paths for these flows. Similar with the failure detection algorithms described in the previous section, there is a recovery approach based on cycle structure [29]. This approach firstly computes a tree for the topology, then assigns a tie-set (in fact a circle) for each remaining link (which is not contained in the tree). As the result, if a link is broken, the algorithm can find and use the responding tie-set to repair.

The paper [24] puts forward a method to recover from link failures in the fat-tree topology where only two switches belonging to different layers can have a connection. If they have a directly connected link, there will going to be at least another detour path consisting of three links to connect them as well. This

Table 2. Comparison of restoration methods

Approaches	[40]	[29]	[24]
Algorithm of computing routes	Shortest path algorithm	Tie-set graph	According to the fat-tree topology
The stability of control messages when recovering	✗	✓	✗
Controllers job (initialization)	Update flow tables for working paths	Update flow tables for working paths and each tie-set	Update flow tables for working paths
Applying to failures of multiple links	✓	✗	✓
Applying to failure of switch	✓	✗	✓
Limitation	✗	✗	Fat tree topology
Consideration of traffic	✗	✗	✓

is the main idea of this approach, and the controller records the load of each link to help decide the protection paths.

At the end of this subsection, a comparison is summarized in Table 2. The controller's job before and after the link failure is discussed. For the second method, as flow table entries for each working path and each tie-set has been worked out before the fault, few packets are needed to change the path from the primary one to the appropriate tie-set, which is contrary to other two approaches.

Next, the scalability problem is explained. For the first and the last method, as long as there is a path available, they can find it for recovering. However, for the second approach, each link only belongs to limited tie-sets, and each tie-set can only deal with a single link failure. As the result, the second approach has drawbacks in expansibility.

4.2 Protection

After the working path is broken, the protection technology can be used which contains two steps (shown in Fig. 5): 1. The controller has worked out the backup path along with the working path, and stored the information into switches' forwarding tables [37], [39] or packet headers [33]. When the failure happens, the switches can detect this fault, and change the route to the backup path without the participation of the controller. 2. Data flows will go through the protection path.

According to the difference in the version of OpenFlow protocol adopted, the methods based on the protection mechanism can be divided into two categories: approaches based on OpenFlow 1.0 and OpenFlow 1.1, respectively. When adopting OpenFlow 1.0, [37] suggests that the working path and backup path can be preserved into two kinds of flow tables which are different in priorities. When the working path breaks, only if its corresponding flow table entries are deleted,

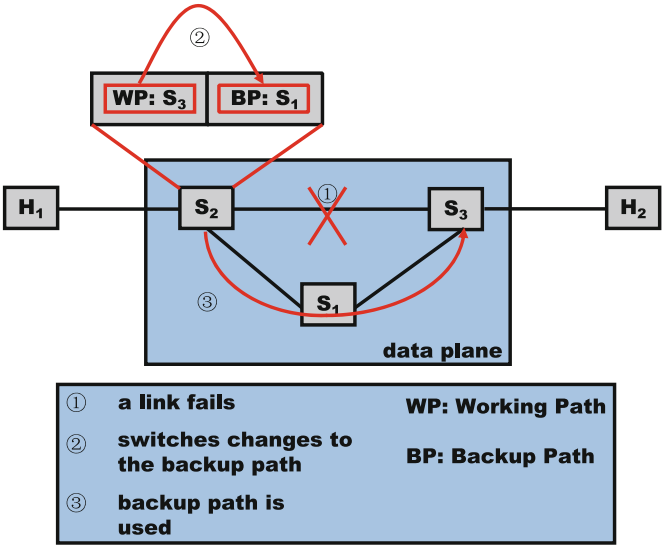

Fig. 5. Protection technology

the protection path which is lower in priority can be used. As the result, this method will generate OFP_FLOW_RESTORE packet to inform the controller to recompute route if the primary one recovers. While [33] proposes the two paths can be coded into the headers of packets. Moreover, a label is assigned for each port of each switch to save space. Nevertheless, the length of route is restricted by the size of the header without doubt.

From OpenFlow 1.1, the concept of group table is proposed, and the method proposed in [39] uses the "fast fail over" group type to save working path and protection path in multiple action buckets.

At the end of this subsection, a comparison is summarized in Table 3. The granularity of protection is discussed which can be divided into two categories: for each link, or for each path. The first two methods compute a backup path for each link contained in the working route. By contrast, the approach based on the group table uses BFD to monitor the whole path periodically. As long as any link of the working path breaks, the whole route is announced to be failed, and the protection path can be activated immediately.

Similar with the restoration, the scalability problem needs to be explained as well. As long as both the working path and the protection path face a failure, data flows can't find another route to deliver packets any more even though there are other routes available.

4.3 Comparisons, Summaries and Open Research Issues

We conducted the simulation experiments on fault tolerance of the data plane. In detail, we implemented a restoration and protection method both using the

Table 3. Comparison of protection methods

Approaches	[37]	[33]	[39]
Method to save routes	Two flow table entries with different priorities	Reuse the packet header	Group table
Method to return to the working path	OFP_FLOW_RESTORE packet	automatically	automatically
Granularity	Each link of the working path	Each link of the working path	The whole working path
Applying to failures of multiple links	×	×	×
Applying to failure of switch	✓	✓	✓
Limitation on the length of each route	×	✓	×
Version of OpenFlow	1.0	1.0	1.1

shortest path algorithm to calculate routes. These two fault tolerance applications were developed on Ryu – a python-based controller [3]. When referring to the topologies, the 4-ary and 6-ary fat-trees [6], and two real-world network scenarios – basic reference topology of the COST 266 action project [26] and a topology from the Topology Zoo [1] – were used for experiments. Specifically, we implemented the protection method by calculating a backup path for each link instead of for each path as the number of links is relatively less and more stable.

When referring to the simulation and emulation tools for SDN, Mininet adopted in our experiments is the most popular one. Besides, [14,36] are especially designed for the distributed experimental environments. By using Mininet [2], we measured three metrics – the number of total table entries, the average number of hops for backup paths, and the recovery time – on each topology. Firstly, as for the recovery time, the protection was much less than the restoration. Moreover, the recovery time of the former was stable as backup paths were calculated in advance and could be activated as soon as links fail. Secondly, the number of table entries was discussed. The ratio between the protection and the restoration was in the range of 2.78 to 3.90. Thirdly, the average number of hops was compared. The backup path calculated by the protection needed 1.23 to 2.53 more hops than the restoration. Specifically, in the fat-tree topology, all the connections have inherent rules. For example, each two hosts have at least two paths of the same length to reach each other, which is proven by the restoration. As the result, a few protection paths for an identical flow can be calculated rapidly according to the inherent rules, and can be stored into the group tables for failure recovery. The benefit is that the recovery time is less than the restoration, and the calculation of protection paths is easy to realize than the algorithms to find disjoint alternative paths (such as Suurballe's algorithm [41]).

Besides the conclusions inferred from the experiments, the comparisons between the two mechanisms can be analyzed in theory. In general, the restoration has the advantage of flexibility. For example, as the size limit of

the group table, it is hard to exhaust all the backup paths before the failures. By contrast, provided that at least one path is available, the restoration can be used for fault tolerance in any case. When considering the metric of length, only the restoration can always compute a shortest path after the failure. Moreover, there are other factors (such as the load of each link) need to be considered except for "the distance of each path" when choosing a backup path. In this situation, the real-time data of each link's load is required, and the restoration can be adopted while the protection cannot satisfy the demand.

As for future research directions, the idea of combination is considered at first. Routing methods in traditional networks can be combined with SDN. As described in [47], the method uses routing tables and flow tables to forward packets before and after link failures, respectively. Another situation is when the controller functions normally, flow tables are used to route messages. Once the controller breaks, all the switches can use traditional routing protocols to calculate paths which looks like a reversion of the former approach. Similarly, there is an idea combining the restoration with protection which aim to take advantages of both.

In the second place, the scalability is of concern as well. Most methods introduced in this paper are focusing on the failure of a single link especially for the protection mechanism, then it can be extended to solve failures of multiple links. As for multiple links, the problem of failure recovery can be further classified according to the properties of these broken links. On one hand, if all the failed links belong to the working path, most existing methods can solve it. In [40], each link is managed by the controller independently. While in [39], no matter how many links are broken in the same working path, the protection path can be active as long as the primary route fails. On the other hand, some failed links belong to the working path, and others are part of the protection path. In this situation, the failure recovery may not be achieved if 1:1 protection is adopted. To solve the limitation, For example, maybe non-intersect paths can be calculated as many as possible. The scalability problem can be extended further to solve the failure of a switch. As this is different from the failure of a link, the difference need to be caught and a more appropriate approach for this situation can be found out. For example, if a switch fails, the controller can judge whether this is a normal fault where just several links break or this is a outage of a switch, then a different solution is given based on different judgment. More specifically, if a port status message of a switch is transmitted to the controller, the controller can probe the other end of this failed link (as the controller knows two switches and their corresponding ports of each link) to confirm whether the opposite switch goes wrong. If so, the controller needs to update the topology and recompute relevant flow table entries.

Moreover, if no backup path is available, the controller can inform the relevant switches to stop transmitting messages walking through these broken links. The problem can be divided into two parts according the types of these links. On one hand, for the link between switches using in-band connection, only a few switches directly connect to the controller. If a node discovers the failure

of a link, it may require a long time to inform the controller. To avoid this phenomena, [12] proposes the switch can inform the affected nodes, other than the controller. The switch on the failed link can use the ingress port to help informing upstream nodes. On the other hand, the link between host and switch is considered. If such a connection breaks, the relevant host can't be reached any more. Therefore the controller need to be informed and relative flow table entries targeting the destination host can be deleted.

There are some papers discussing the problem of fault tolerance from different angles. [34] proposes a declarative language for network administrators to specify working paths as well as backup paths. Based on the syntax and the normalization rules of this language called FatTire, a program can be generated and a compiler is used for translating into configurations of switches. [46] discusses the topic of multicast in SDN, but the fault tolerance problem has been left out which needs to be considered as well.

5 Fault Tolerance on Control Plane

To guarantee the resiliency of the control plane, there are two aspects need to be considered. Firstly, the controller must function properly, which means that the failure of the controller is not allowed. Secondly, switches can communicate with the controller even though the OpenFlow channel breaks. In this section, these two aspects are introduced respectively.

On one hand, considering the situation where the controller is down, the OpenFlow channels are useless, and the underlying switches are out of control. A general solution (as shown in Fig. 6) is using multiple controllers to provide

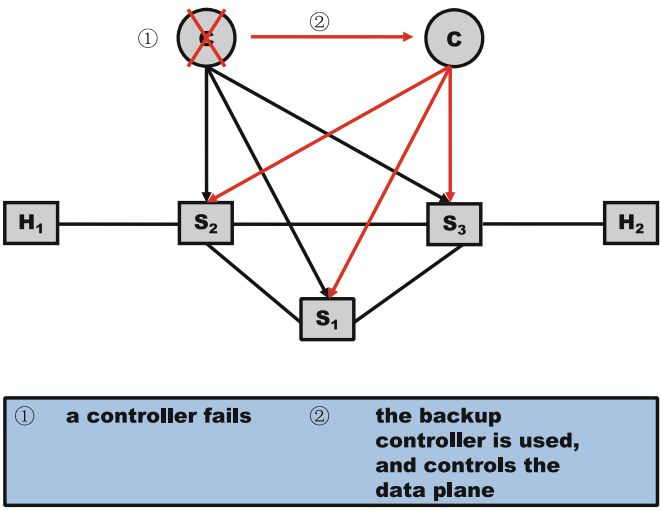

Fig. 6. Deal with the failure of controller

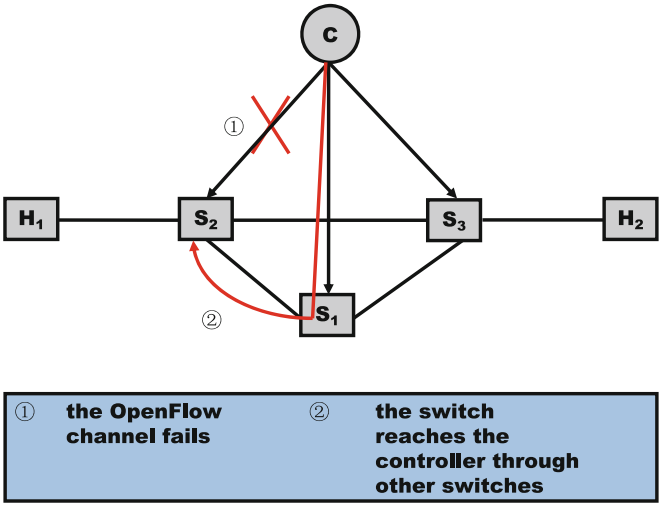

Fig. 7. Deal with the failure of OpenFlow channel

resiliency. When the primary one is wrong, the backup controller can take over the whole data plane.

The problem mentioned above can be further divided into two parts – whether one controller or multiple controllers are in use. On one hand, if only one controller is adopted, its failure is catastrophic. As long as the network changes after the controller fails, there is going to be inconsistency between the data plane and control plane. To solve this problem, for example, traditional routing protocols can take over the subsequent duties to manage the network. Moreover, considering the situation that the data plane seldom goes wrong, the protection mechanism is more recommended as it can operate without the controller even though failures happen. On the other hand, if multiple controllers are applied, the control plane is far more stable than the former. However, in this situation, consistency need to be guaranteed [23]. For example, when the primary controller fails, the left controllers can take over the responsibility. [10,30] are focusing on this problem. The former proposes using a backup controller to avoid a single point of failure. While the latter suggests that besides multiple controllers being used at the same time, a data store can be adopted to ensure the consistency of these managers. As for each switch, it connects to two controllers simultaneously – one is the master, and the other is the slave. Instead of using a data store, [42] suggests adopting a distributed file system to coordinate between controllers. More than just considering how to recover the controllers' states, maintaining switches' states should be involved as well. [16] has achieved this goal by processing the whole event-processing cycle (including: generation of events by switches, communication between switches and controllers, and reconfiguration of relevant switches) as a transaction.

On the other hand, considering the situation where the OpenFlow channel breaks with other network elements operating normally. In general, the out-band and in-band control can be adopted before and after failures, respectively (as shown in Fig. 7). When the control traffic goes through the same path as the data flow does, the fault tolerance methods on the data plane mentioned before can be used for reference [38].

6 Conclusion

SDN is a novel network architecture which decouples the data plane from the control plane. However, SDN is not able to recover from failures automatically. Therefore mechanisms need to be designed for fast failover via the coordination between controller and switches. In this paper, we broadly discuss the fault tolerance issue and systematically review the existing methods proposed so far for SDN. Our representation starts from the significant components that OpenFlow and SDN brings – which are useful for the purpose of failure recovery, and is then further expanded to the discussion of fault tolerance in data plane and control plane, in which two phases – detection and recovery – are both needed. In particular, as the important part of this paper, we have highlighted the comparison between two main methods – restoration and protection – for failure recovery. So far, the recovery for multiple link failure remains an open issue and could be our future research direction. As one of the first few authors, we have shed a light in this paper on the issue of fault tolerance for software defined networks. We expect that our work could attract more researchers' attention, and encourage them to deliver more contributions in this issue.

Acknowledgments. Corresponding authors: Jinbang Chen and Fei Xu. They are with Shanghai Key Laboratory of Multidimensional Information Processing & Department of Computer Science and Technology, East China Normal University, China. This work was supported by the Science and Technology Commission of Shanghai Municipality under research grant no. 14DZ2260800, and China Postdoctoral Science Foundation under grant no. 2014M561438.

References

1. The internet topology zoo. http://www.topology-zoo.org/
2. Mininet. http://mininet.org/
3. Ryu. http://osrg.github.io/ryu/
4. Openflow switch specification: version 1.0.0, December 2009
5. Openflow switch specification: version 1.1.0, Feburuary 2011. http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf
6. Al-Fares, M., Loukissas, A., Vahdat, A.: A scalable, commodity data center network architecture. In: 2008 ACM International Conference on Special Interest Group on Data Communication (SIGCOMM), pp. 63–74, August 2008

7. Atlas, A.K., Zinin, A., Torvi, R., Choudhury, G., Martin, C., Imhoff, B., Fedyk, D.: Basic specification for IP fast reroute: loop-free alternates. In: RFC-5286, September 2008. https://tools.ietf.org/html/rfc5286
8. Basu, A., Riecke, J.: Stability issues in OSPF routing. In: 2001 ACM International Conference on Special Interest Group on Data Communication (SIGCOMM), pp. 225–236, August 2001
9. Bonaventure, O., Filsfils, C., Francois, P.: Achieving Sub-50 milliseconds recovery upon BGP peering link failures. IEEE/ACM Trans. Netw. **15**(5), 1123–1135 (2007)
10. Botelho, F.A., Ramos, F.M.V., Kreutz, D., Bessani, A.N.: On the feasibility of a consistent and fault-tolerant data store for sdns. In: 2013 2nd European Workshop on Software Defined Networks (EWSDN), pp. 38–43, October 2013. http://dx.doi.org/10.1109/EWSDN.2013.13
11. Bryant, S., Previdi, S., Shand, M.: A framework for IP and MPLS fast reroute using not-via addresses. In: RFC-6981, August 2013
12. Desai, M., Nandagopal, T.: Coping with link failures in centralized control plane architectures. In: 2010 2nd International Conference on Communication Systems and NETworks (COMSNETS), pp. 79–88, January 2010. http://dl.acm.org/citation.cfm?id=1831443.1831452
13. Farhady, H., Lee, H., Nakao, A.: Software-defined networking: a survey. Comput. Netw. **81**, 79–95 (2015)
14. Ficco, M., Avolio, G., Palmieri, F., Castiglione, A.: An HLA-based framework for simulation of large-scale critical systems. Concurr. Comput.: Prac. Exp. (2015). doi:10.1002/cpe.3472
15. Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J., Hölzle, U., Stuart, S., Vahdat, A.: B4: experience with a globally-deployed software defined wan. In: 2013 ACM International Conference on Special Interest Group on Data Communication (SIGCOMM), pp. 3–14, August 2013
16. Katta, N., Zhang, H., Freedman, M., Rexford, J.: Ravana: controller fault-tolerance in software-defined networking. In: 2015 1st ACM SIGCOMM Symposium on Software Defined Networking Research, pp. 4:1–4:12, June 2015
17. Katz, D., Ward, D.: Bidirectional forwarding detection. In: RFC-5880, June 2010
18. Kim, H., Santos, J.R., Turner, Y., Schlansker, M., Tourrihes, J., Feamster, N.: Coronet: fault tolerance for software defined networks. In: 2012 20th IEEE International Conference on Network Protocols (ICNP), pp. 1–2, October 2012
19. Kozat, U.C., Liang, G., Kokten, K.: On diagnosis of forwarding plane via static forwarding rules in software defined networks. In: 2014 33rd IEEE Conference on Computer Communications (INFOCOM), pp. 1716–1724, April 2013. http://arxiv.org/abs/1308.4465
20. Kreutz, D., Ramos, F., Esteve Rothenberg, P., Esteve Rothenberg, C., Azodolmolky, S., Uhlig, S.: Software-defined networking: a comprehensive survey. Proc. IEEE **103**(1), 14–76 (2015)
21. Lee, S., Yu, Y., Nelakuditi, S., Zhang, Z.L., Chuah, C.N.: Proactive vs. reactive approaches to failure resilient routing. In: 2004 23rd IEEE Conference on Computer Communications (INFOCOM), pp. 176–186, March 2004. http://arxiv.org/abs/1308.4465
22. Lee, S., Li, K.Y., Chan, K.Y., Lai, G.H., Chung, Y.C.: Path layout planning and software based fast failure detection in survivable openflow networks. In: 2014 10th International Conference on the Design of Reliable Communication Networks (DRCN), pp. 1–8, April 2014

23. Levin, D., Wundsam, A., Heller, B., Handigol, N., Feldmann, A.: Logically centralized? state distribution tradeoffs in software defined networks. In: 2014 Proceedings of 3rd Workshop on Hot Topics in Software Defined Networking (HotSDN), pp. 1–6, January 2012
24. Li, J., Hyun, J., Yoo, J.H., Baik, S., Hong, J.K.: Scalable failover method for data center networks using openflow. In: 2014 14th IEEE Network Operations and Management Symposium (NOMS), pp. 1–6, May 2014
25. Liu, Z., Li, Y., Su, L., Jin, D., Zeng, L.: M2cloud: software defined multi-site data center network control framework for multi-tenant. In: 2013 ACM International Conference on Special Interest Group on Data Communication (SIGCOMM), pp. 517–518, August 2013
26. Maesschalck, S., Colle, D., Lievens, I., Pickavet, M., Demeester, P., Mauz, C., Jaeger, M., Inkret, R., Mikac, B., Derkacz, J.: Pan-european optical transport networks: an availability-based comparison. Photonic Netw. Commun. **5**(3), 203–225 (2003). http://dx.doi.org/10.1023/A%3A1023088418684
27. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: Openflow: enabling innovation in campus networks. ACM Comput. Commun. Rev. **38**(2), 69–74 (2008)
28. Moy, J.: OSPF version 2. In: RFC-2328, April 1998
29. Nagano, J., Shinomiya, N.: A failure recovery method based on cycle structure and its verification by openflow. In: 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), pp. 298–303, March 2013
30. Fonseca, P., Bennesby, R., Mota, E., Passito, A.: A replication component for resilient openflow-based networking. In: 2012 IEEE 13th Network Operations and Management Symposium (NOMS), pp. 933–939, April 2012
31. Przygienda, T., Shen, N., Sheth, N.: M-ISIS: multi topology (MT) routing in intermediate system to intermediate systems (IS-ISs). In: RFC-5120, February 2008
32. Psenak, P., Mirtorabi, S., Roy, A., Nguyen, L., Pillay-Esnault, P.: Multi-topology (MT) routing in OSPF. In: RFC-4915, June 2007
33. Ramos, R.M., Rothenberg, C.E.: Slickflow: resilient source routing in data center networks unlocked by openflow. In: 2013 IEEE 38th Conference on Local Computer Networks (LCN), pp. 606–613, October 2013
34. Reitblatt, M., Canini, M., Guha, A., Foster, N.: Fattire: declarative fault tolerance for software-defined networks. In: 2013 Proceedings of 2nd Workshop on Hot Topics in Software Defined Networking (HotSDN), pp. 109–114, August 2013
35. Rongqing, C.: Research on the fast failure recovery technologies of IP networks. Master's thesis, Hangzhou Dianzi University, March 2012
36. Roy, A.R., Bari, M.F., Zhani, M.F., Ahmed, R., Boutaba, R.: Dot: distributed openflow testbed. In: 2014 ACM International Conference on Special Interest Group on Data Communication (SIGCOMM), pp. 367–368, August 2014
37. Sgambelluri, A., Giorgetti, A., Cugini, F., Paolucci, F., Castoldi, P.: Openflow-based segment protection in ethernet networks. IEEE/OSA J. Opt. Commun. Netw. **5**(9), 1066–1075 (2013)
38. Sharma, S., Staessens, D., Colle, D., Pickavet, M., Demeester, P.: Fast failure recovery for in-band openflow networks. In: 2013 9th International Conference on the Design of Reliable Communication Networks (DRCN), pp. 52–59, March 2013
39. Sharma, S., Staessens, D., Colle, D., Pickavet, M., Demeester, P.: Openflow: meeting carrier-grade recovery requirements. Comput. Commun. **36**(6), 656–665 (2013). http://www.sciencedirect.com/science/article/pii/S0140366412003349

40. Staessens, D., Sharma, S., Colle, D., Pickavet, M., Demeester, P.: Software defined networking: meeting carrier grade requirements. In: 2011 18th IEEE Workshop on Local Metropolitan Area Networks (LANMAN), pp. 1–6, October 2011
41. Suurballe, J.W.: Disjoint paths in a network. Networks **4**(2), 125–145 (1974)
42. Tootoonchian, A., Ganjali, Y.: Hyperflow: a distributed control plane for openflow. In: 2010 7th Internet Network Management Conference on Research on Enterprise Networking (INM/WREN), p. 3, April 2010
43. Vasseur, J.P., Pickavet, M., Demeester, P.: Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS. Morgan Kaufmann, San Francisco (2004)
44. Wang, S., Li, D., Xia, S.: The problems and solutions of network update in SDN: a survey. In: 2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 474–479, April 2015
45. Wei, T., Mishra, P., Wu, K., Zhou, J.: Quasi-static fault-tolerant scheduling schemes for energy-efficient hard real-time systems. J. Syst. Softw. **85**(6), 1386–1399 (2012)
46. Gu, W., Zhang, X., Gong, B., Wang, L.: A survey of multicast in software-defined networking. In: 2015 5th International Conference on Information Engineering for Mechanics and Materials (ICIMM), July 2015
47. Yu, Y., Shanzhi, C., Xin, L., Yan, W.: A framework of using openflow to handle transient link failure. In: 2011 1st International Conference on Transportation, Mechanical, and Electrical Engineering (TMEE), pp. 2050–2053, December 2011