Revive: A Reliable Software Defined Data Plane Failure Recovery Scheme

Israat Haque
Computer Science, Dalhousie University
israat@dal.ca

MA Moyeen
Computer Science, Dalhousie University
mamoyeen@dal.ca

***Abstract*—In Software-Defined Networking (SDN) links and switches (nodes) from data plane suffer from failure and impact network operations. In the presence of such failures, switches can use *reactive* or *proactive* recovery scheme. In the reactive scheme, switches contact the controller after detecting a link failure to get a backup route setup; whereas, switches locally redirect the traffic to the backup route without controller's intervention in the case of the proactive scheme. In this paper, we propose a hybrid recovery scheme, called *Revive*, where the controller proactively installs backup routes only in a subset of the switches between a source-destination pair. In addition, we judicially configure the routes in Revive to meet the application and the reliability demand while efficiently utilize the network resources. Extensive experimental results in Mininet using real topologies illustrate the benefits of Revive compared to its counterparts.**

I. Introduction

Software-Defined Networking (SDN) [1] removes network control functions from hardware like routers and switches, which in turn makes these devices simple packet forwarders. The SDN architecture is composed of management, control, and infrastructure planes, where the policy of the network services or applications is defined at the management layer. The control plane (controller) then translates this policy into network configuration rules for the infrastructure layer (data plane) devices. The management and control layers communicate through a Northbound API; whereas, the control and data plane communicates using the Southbound (SB) API. The control plane installs packet forwarding rules in the flow tables of the software enabled data plane elements using the de facto SB API OpenFlow [2].

SDN improves the flexibility and fosters the innovation in network monitoring, management and operations. Thus, SDN-based network design is adopted in data centres, enterprises, cloud and cellular networks [3], [4]. Also, SDN has received attention in other emerging networking applications like the Internet of Things (IoT) and smart cities [5], [6]. All these applications may suffer from service interruption because of the failure of links and devices from the data plane. For instance, cloud network providers suffered from 1600 hours of service disruptions and US$273 million loss from 2007 to 2013 due to application and infrastructure failures [7].

There are two failure recovery approaches; namely *reactive* (*restoration*) and *proactive* (*protection*) [8], [9]. In the reactive approach, upon detecting a link failure switches contact the controller to get the backup route set up to redirect the affected traffic. Thus, a delay is associated with the reactive approach to recover from the link failure. On the other hand, in the case of proactive approach controller preinstalls backup routes at the flow tables of the switches. The switches upon detecting a link failure locally redirect the affected traffic without the controller's intervention. Thus, the reactive scheme incurs delay without any burden on the limited memory (TCAM) of the OpenFlow switches; whereas, the proactive scheme reduces that delay at the price of extra TCAM usage.

In this paper, we propose a hybrid link failure recovering scheme called *Revive*, which combines the benefits from both the reactive and proactive approaches. Revive preinstalls backup routes at a subset of the switches between a given source-destination pair, where switches locally redirect the affected traffic. However, the success of Revive or any other recovery schemes will depend on the availability of backup routes. Thus, we must carefully design a routing topology that guarantees backup routes between every pair of switches. The benefits of such topology construction include available backup routes, congestion avoidance, and load distribution [10], [11].

In the Revive architecture, the controller constructs and maintains a k-edge-disjoint routing topology based on the algorithm proposed in [12]. The strength of this topology construction algorithm is that in addition to guaranteeing disjoint routes; it allows the network operators or designers to select appropriate topology structure to meet the demand of the applications or services. For instance, we can choose structures like Shortest Path Tree (SPT), Minimum Spanning Tree (MST), or Degree Constraint SPT (DCSPT). Each structure has unique properties suitable for different services. Finally, this precomputed disjoint routes can reduce the backup route installation time even for the reactive scheme as the controller will not need to compute the backup route upon receiving a path restoration request.

We further observe that there is a tradeoff between the network policy implementation and the efficient resource utilization at the data plane switches. For instance, the network operators may need to establish the shortest routes among the switches, which may heavily utilize the TCAM of the switches along the chosen shortest routes. We exploit the k-edge disjoint routing topology, the global network view of the controller, the network traffic analysis, and the dynamic network programmability of the SDN architecture and propose

978-3-903176-14-0

a novel solution to utilize the TCAM efficiently. We observe that the route-stretch (the ratio of the actual to the optimal routes) of the disjoint secondary paths of the routing topologies is close to their optimal values. Thus, we can switch to the disjoint secondary route after the memory usage at the switches along the primary route reaches a threshold to improve the memory utilization and throughput without violating the application's policy. In addition, our initial experimental results reveal that SPT and MST offer a similar route-stretch, throughput, restoration delay, and memory usage. Thus, we may even interchangeably use these structures to distribute the memory usage among switches.

We implement Revive and compare against [13], [8] in Mininet [14] emulator. We use Ryu controller [15] and Open vSwitch (OVS) [16], where OVSs use OpenFlow 1.3 to communicate with the Ryu controller. We also use OpenFlow's Fast Failover Group (FFG) [17] to locally redirect the affected traffic. Extensive experiments on real network topology demonstrate that Revive significantly improves both the path restoration delay and TCAM usage compared to its counterparts. In addition, Revive balances the memory utilization among the switches while guarantees reliability and meets the application policy.

Contributions: The contribution of this paper can be summarized as follows;

- Design a hybrid link failure restoration scheme called Revive to improve the restoration delay and memory usage.
- Propose a novel approach of distributing the memory usage among switches while guaranteeing reliability and meet the application policy.
- Conducted extensive experiments to evaluate Revive using real network topology.

The remaining of the paper is structured as follows. We compare and contrast the work related to Revive in Section II. We present the detailed design of Revive in Section III. The experimental setup and associated parameters are presented in Section IV. We present and discuss the experimental results in Section V, which follows concluding remarks at Section VI.

II. Related Work

In this section compare and contrast existing proactive and reactive solutions related to Revive. The proactive recovery scheme installs backup routes at the switches to locally detect and recover from link failures. Thus, these schemes reduced the delay by avoiding communication to the controller at every failure instances. The authors in [18] proposed a proactive recovery scheme using alternate routes, flow rules compression, source routing, and network virtualization. However, the solution requires the packet headers to carry extra routing information, which may have an impact on the scalability.

The work [19] used an enhanced Breadth First Search (BFS) based proactive backup path installation scheme. A couple of works in [20], [21] use Bidirectional Flow Detection (BFD) [22] protocol and FFG; whereas, the work [23], [24] used Link Layer Discovery Protocol (LLDP) and VLAN tags to locally detect and recover from a link failure. These works have not considered distributing memory utilization among the switches, which is a key contribution of Revive. The authors in [8], [25] locally detected congestion at the switches and redirected the affected traffic towards the backup routes, which may create oscillation across the network. The work in [26] proposed a controller-managed congestion mitigation scheme without distributing memory utilization.

Reactive link failure recovery schemes need to contact the controller upon detecting a failure. Thus, these schemes incur communications delay while efficiently utilize the TCAM. A reactive link failure recovery method is proposed in [27], where a failed link is detected with continuous heartbeat messages. The scheme in [13] optimized the path cost in terms of the number of hops while reactively selected the backup routes. A similar reactive recovery approach is presented in [28]. In contrast to the above proactive and reactive schemes Revive attempts to bring the best from these two approaches. Revive proactively installs flow rules only for a subset of switches that carry traffic for the ongoing communications.

Chen *et al.* divided the TCAM into two, where flow rules are installed in a small table to reduce the flow installation time and efficient TCAM usage [29]. The work in [18], [30] proposed to compress flow rules that share same actions and output ports. A TCAM aware routing is proposed in [31] to reduce the memory demand. These works could be complementary solutions for Revive, where the goal is to distribute the TCAM usage among the switches without violating application policy.

III. Revive Architecture and Design

The architecture of Revive is shown in Figure 1. The demand or policy from services and applications are defined at the application layer. Thus, the routing policy of Revive is defined at the application layer. The controller acquires that policy to define the route configuration rules for the OpenFlow enabled data plane elements. Furthermore, the

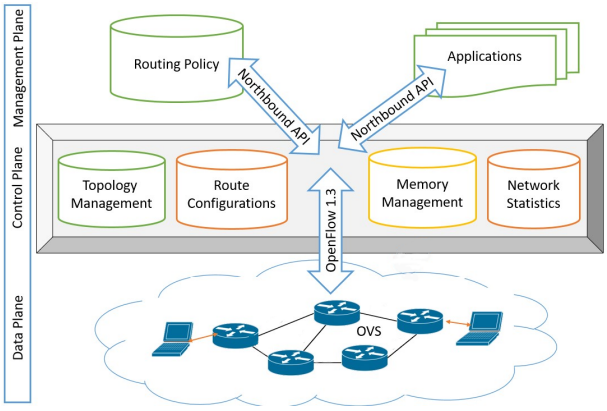


Fig. 1. Revive Architecture.

controller periodically gathers the network state information to dynamically update the network configuration rules to meet the demand of the applications. In the following, the operation of the major modules of Revive is presented.

Topology management: This module collects and maintains the information of the data plane elements to define the network topology. We define the network topology as a graph $G(V, E)$, where V is the set of data plane switches and E is the set of links among them. We then defined a k-edge connected *routing topology* $D_k(G)$ of $G(V, E)$ such that there are k edge-disjoint alternative routes between every pair of nodes in $D_k(G)$. We use the algorithm in [12] to construct the k-edge connected routing topology using spanning structures SPT or MST. In the case of any link failure, the affected switch sends a control message to the Revive controller to update the network topology. In addition, the controller periodically gathers the network state information to update the topology.

Route configuration: The hybrid route configuration scheme of Revive is depicted in Algorithm 1. The algorithm states that upon receiving a flow installation request from a switch, the controller uses Algorithm 1 to determine the primary (optimal) and backup (near-optimal) routes for the given source-destination pair. The controller then installs both the primary and backup links at each switch along only the primary route. Recall that in the reactive recovery scheme controller installs the flow rule after receiving an installation request, which may occur because of regular packet generation from a host or because of a link failure. On the other hand, proactive scheme preinstalls all possible primary and backup routes at each switch without considering their demand.

Algorithm 1 Flow Installation Algorithm

1: **Input**: sr, dt
2: **Output**: $PrimaryPort_i$, $SecondaryPort_i$
3: $SW_p \leftarrow |PrimaryRoute(sr, dt)|$
4: **for** $i \in SW_p$ **do**
5: $l_p \leftarrow PrimaryRoute(i, dt)$
6: $l_b \leftarrow SecondaryRoute(i, dt)$
7: $FlowTable_i \leftarrow l_p$
8: $PrimaryPort_i \leftarrow ExtractPort(l_p)$
9: $SecondaryPort_i \leftarrow ExtractPort(l_b)$
10: $ConfigureFFG(PrimaryPort_i, SecondaryPort_i)$
11: **end for**

In Algorithm1, sr, dt, and SW_p define a source, a destination, and the number of switches along the primary route between sr and dt, respectively. For each of these switches, Revive installs the primary route in the associated flow tables. It also configures the FFG group of these switches with the primary and secondary ports. Thus, initially both the reactive and Revive starts with a setup of switches having no pre-installed flow rules in their TCAM. However, Revive installs flow rules according to Algorithm 1 upon receiving a flow installation request. If there are t source-destination pairs having k alternate routes with an average primary route length of L, then the memory complexity of Revive can be $O(ktL)$; whereas, the memory complexity of the proactive approach would be $O(ktLV)$. Thus, Revive significantly reduces the TCAM demand without incurring extra communication delay in the presence of a link failure.

Detecting link failure and redirecting traffic: In Revive, we use OpenFlow's FFG to detect a link failure at the data plane and redirects the traffic to the available alternate route. The FFG consists of a list of buckets, each with associated actions and *watch port* (*watch group*) to monitor the "liveliness" of a port (group). At a time a single live port or group is active. In the case of a failure, we can choose the next available live port or group. FFG can use BFD protocol to detect the link status, where a connected pair of switches exchanges periodical HELLO messages to measure the "liveliness" of the associated link. Any missing HELLO message for a certain period indicates that the associated link is down.

Memory management: The task of this module is to optimize the TCAM usage at the switches and the route-stretch without violating the application policy. Suppose M_{tcami} represents the TCAM occupancy of a switch i between a source-destination pair with a route-stretch L_{rtj}. The memory management module optimizes the following objective function.

$$min \quad \alpha \sum_{i=1}^{|L|} M_{tcami} + \beta L_{rtj} \tag{1}$$

Here α and β are the controlling parameters to give appropriate weight to the memory usage or route-stretch as per the demand of the application. We use a heuristic to balance between the memory utilization and the route-stretch, which is outlined below.

The memory management module first installs the flow rules along the primary route between a source-destination pair as per the application policy. It then monitors the memory usage along that route and switches to the backup route once the memory usage of a switch along the primary route reaches a predefined threshold. Furthermore, this module can dynamically update the weight to balance between the reliability, the efficient resource utilization, and the application demand. The proposed routing topology guarantees that the route-stretch of the backup routes is close to the optimal one. Thus, by switching to the backup routes, the memory utilization is distributed among the switches without compromising on the route-stretch and the application demand.

IV. Experimental Setup

This section describes the emulation environment used for the evaluation of Revive. We run the emulation on a server

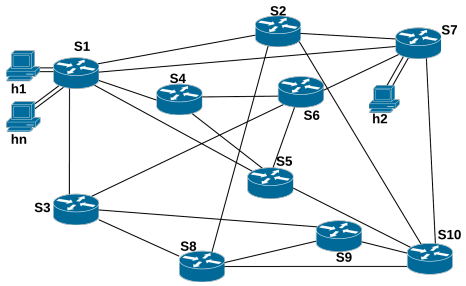


Fig. 2. A ten node topology.

with 2.66GHz CPU, 45GB RAM, and 12 cores. The Mininet 2.3 emulator runs on the Ubuntu 16.04.3 LTS operating system. Open vSwitch 2.9.2 kernel switch is used to emulate the switches at the data plane. These switches use OpenFlow 1.3 protocol to communicate with the Ryu 4.24 controller over links with a bandwidth of 1.5Mbps. We consider a ten node

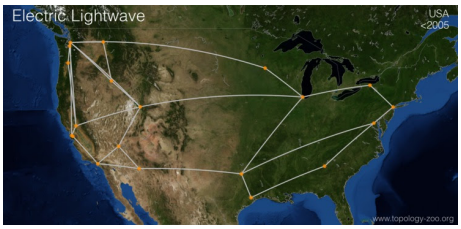


Fig. 3. A twenty node carrier-grade network topology from Electric Lightwave.

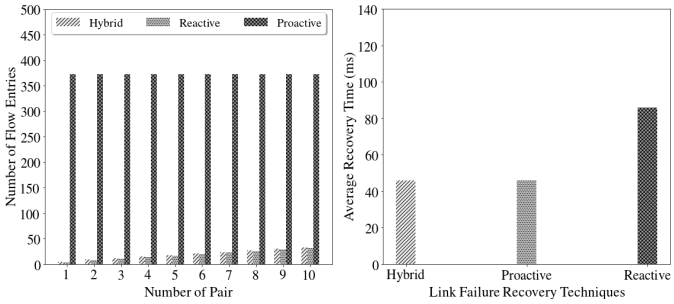
topology shown in Figure 2 and a twenty node carrier-grade network topology from the Electric Lightwave [32] shown in Figure 3 to evaluate the performance of the proposed solution. We randomly choose ten source-destination pairs to route traffic among them. They are chosen from the two opposite edges of the topology to force the traffic to flow through a long route. We capture the traffic at appropriate ports using tcpdump.

We measure the link failure recovery time, the TCAM usage, the throughput, and the hop-stretch (ratio of the number of hops between a source-destination pair to the optimal number of hops) to evaluate the performance of Revive, where the results are the average of ten pairs. The 95% confidence interval is quite small; thus, omitted from the figures.

The reaction time is calculated based on the *timestamp* of the packets while traversing through the ports and the links. For instance, the difference between the timestamps at an exit and an entry port can be defined as the recovery time. The link failure is detected using the BFD protocol at the switches. We consider multiple link failures with a different percentage and measure corresponding recovery time and hop-stretch. In particular, we randomly fail a link between a given source-destination pair; thus, switch to the backup route. In the backup route we again randomly fail another link. We continue this process until we reach the destination. To measure the throughput, we vary the traffic loads at the sources and use iPerf [33] utility. Furthermore, we vary the number of hosts at the switches while measuring the throughput. We consider both the TCP and UDP traffic. We construct a 2-edge connected routing topology by first extracting a subgraph, $D_1(G)$ (based on SPT or MST), of the physical topology $G(V, E)$. We then remove the edges of $D_1(G)$ from G, which may make G composed of multiple connected components. We build second 1-edge connected subgraph out of these components, and merge them with $D_1(G)$ to form $D_2(G)$.

V. Experimental Results and Discussion

In this section, we first present the experimental results to evaluate and compare Revive with reactive [13] and proactive [8] approaches in terms of the TCAM usage and the recovery time from link failures. Next, we present the hop-stretch of these three schemes with different percentage of link failures. The performance of SPT and MST based routing topologies then come regarding their hop-stretch and throughput. Finally, we present the results on distributing the TCAM usage among switches to enhance the throughput while offering the reliability and meet the application policy. We consider real topology from a carrier-grade network to evaluate the performance of Revive in addition to studying an emulated topology.



(a) TCAM usage (b) Recovery time

Fig. 4. The average TCAM usage and the link failure recovery time in Mininet topology.

Evaluation of memory usage and failure recovery time: Figure 4 presents the TCAM usage and the average link failure recovery time of reactive, proactive, and hybrid (Revive) schemes on SPT-based routing topology. The TCAM usage of the reactive scheme is the best as it installs on-demand flow rules only after receiving a packet at a source or after experiencing a link failure. Revive installs the same number of flows in the TCAM of the switches along the primary route; however, it installs primary and backup ports in the associated group table. In addition, Revive needs a

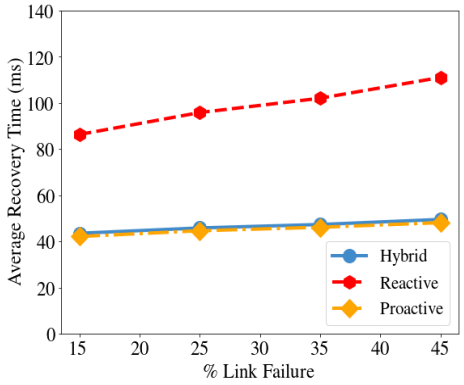
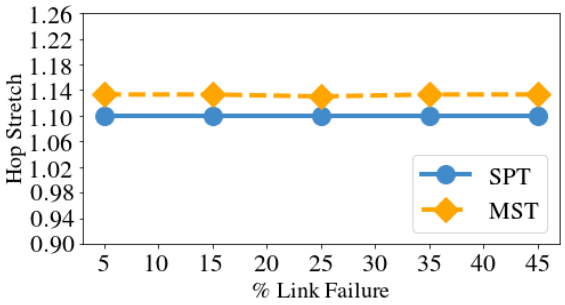


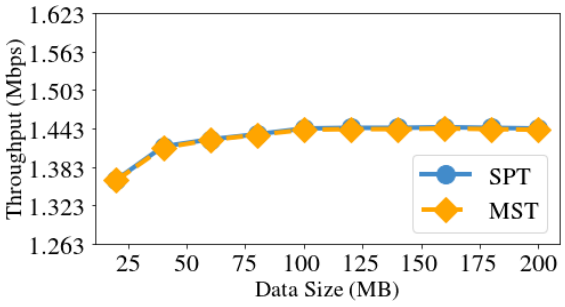
Fig. 5. The average link failure recovery time in Mininet topology.

couple more flow entries along the primary route to switch over to the backup route once when TCAM usage exceeds a threshold. The proactive approach preinstalls all possible flow rules in all switches; thus, consumes the highest amount of memory. The number of flow entries increases with increasing number of source-destination pairs for both the reactive and Revive; whereas, the TCAM uses of proactive approach remains constant irrespective of the change in the number of

pairs. The link failure recovery time of the proactive scheme is the best because of the available rules in all switches. Thus, the switch can use FFG to locally redirect the traffic without communicating with the controller. The reactive scheme has the worst recovery time as it needs to contact the controller for every single failure events. The recovery time of Revive is similar to that of proactive approach as Revive avoids reaching the controller at every failure. Figure 5 illustrates the trend of the average link failure recovery time of the proactive, reactive, and hybrid schemes with different percentage of link failures. Unlike the correlated multiple link failures used in [34], we consider multiple disjoint link failures. In particular, for a given source-destination pair we recursively fail a link along the chosen route until a packet reaches the destination. The reactive approach has the worst recovery time that increases with the increasing percentage of link failure. This behavior is expected as switches need to contact the controller for the backup route setup. The recovery time of the proactive and Revive is almost constant irrespective of the percentage of link failure.



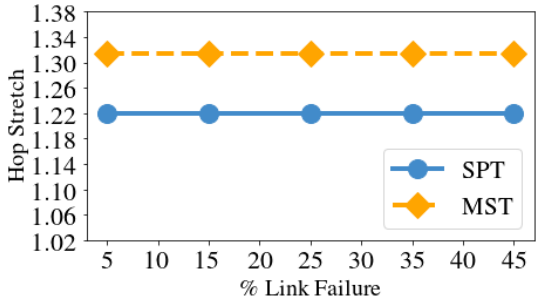
(a) Hop-stretch



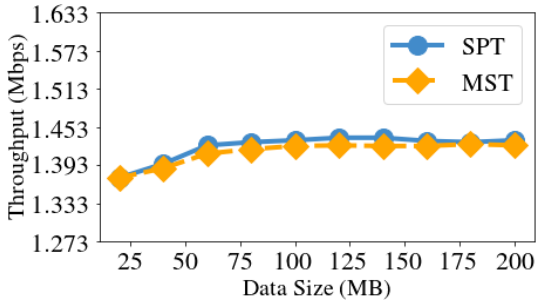
(b) Throughput

Fig. 6. The average hop-stretch and the throughput in Mininet topology.

Impact of routing topologies: We will next investigate the performance of Revive on various routing topologies. This evaluation will be useful for the services or applications to select the right routing topology while configuring routes. We consider SPT and MST as the candidate routing topologies; however, any spanning structures can be used. We have considered a different percentage of link failures and traffic loads while measuring the average hop-stretch and the throughput of Revive on SPT and MST in Mininet topology (Figure 6). The SPT has the better hop-stretch compared to the MST as the former one follows the shortest route. MST usually selects shorter links thus experiences a few extra hops. However, this might be useful to distribute loads among the switches, which we observe while measuring their throughput. Their throughput is quite similar. We then estimate the same hop-



(a) Hop-stretch



(b) Throughput

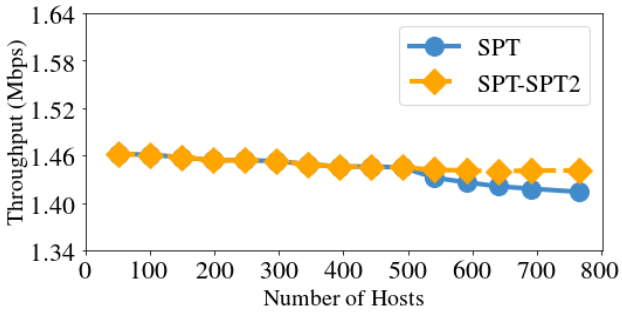
Fig. 7. The average hop-stretch and the throughput in real topology.

stretch and throughout of these two routing topologies on real network topology (Figure 7) from a carrier-grade network provider called Electric Lightwave from the USA. The result is similar to the one we observed with the emulated topology.

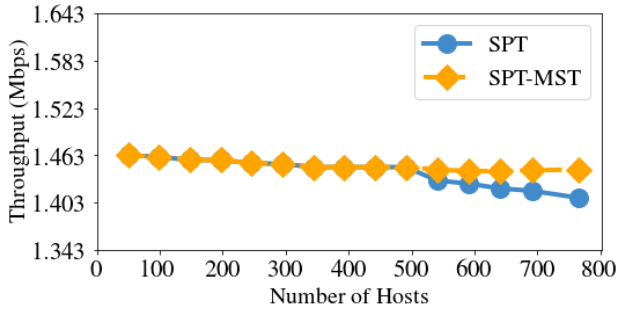
Consolidating memory usage: In applications like data centers and cloud networks, hosts may dynamically join the networks. If the application policy demands the shortest routes, then the TCAM of the switches along the primary route will soon be overwhelmed by the newly joined hosts. However, the TCAM of the switches from the backup routes might be underutilized. The hop-stretch of these backup routes of SPT and MST does not deviate significantly from the optimal value of 1.0. In addition, their throughput and link failure recovery time is quite impressive. Thus, we next focus on consolidating the TCAM usage of the switches using these spanning structures while supporting the reliability, enhancing the throughput and meeting the application policy.

In terms of distributing the memory usage, the controller in Revive first configures the primary and backup routes as per the application policy. It then monitors the TCAM usage of the switches and delegates the routing to the backup routes upon detecting that the memory usage of a switch along the primary route exceeds a predefined threshold. Without delegating the routing to the backup routes, TCAM along the primary route will be full, and switches will start dropping packets. We have considered both the SPT and the MST based routing topologies, where packets initially follow the SPT/MST based primary route, then switch to the secondary route of the re-

spective topology, i.e., SPT2/MST2 (disjoint secondary route). It is also possible to combine these two routing topologies; e.g., SPT based primary and MST based secondary or vice versa. The evaluation results in Mininet topology is shown in



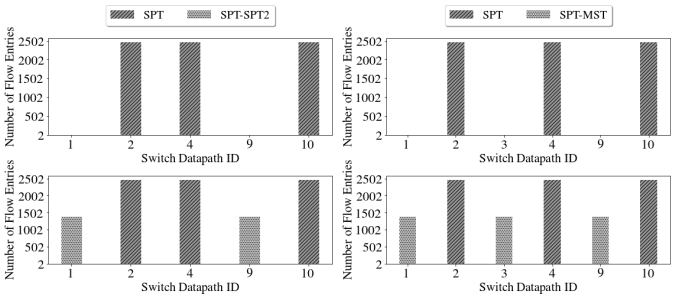
(a) SPT based



(b) SPT-MST based

Fig. 8. The average throughput in Mininet topology.

Figure 8. It is apparent from the results that better throughput can be achieved by consolidating the memory usage among the switches from the primary and the secondary routes instead of strictly using the former one. In particular, without delegating the TCAM usage Revive will continue to use the primary route and start dropping packets when the TCAM gets full. We consider SPT and SPT-MST based topology to distribute the TCAM usage. In the former case, both the primary and the backup routes are SPT based; while in the latter case the primary route is SPT based and the backup one is MST based. In both cases, we observe similar performance with the fact that the throughput can be improved by distributing the memory usage. Figure 9 illustrates the TCAM usage among



(a) SPT based (b) SPT-MST based

Fig. 9. The average TCAM utilization in Mininet topology.

the switches from the primary and the backup routes of the chosen SPT and SPT-MST based topology. The top plot in each sub-figure shows the memory distribution among the switches between a source-destination pair, where the TCAM of the switches from the backup routes is not used. These available memories can be used to better distribute the memory utilization and to enhance the throughput, which is shown in the bottom plots of the sub-figures. We use VLAN tagging at the source to split the traffic among the primary and secondary routes; whereas, intermediate switches install actual flow rules. We furthermore measure the throughput, and the TCAM uses on the real topology and obtains similar results that are shown in Figure 9. However, we do not include those results due to the space limitation.

Take away lessons: It is always useful to bring together the best services from different design approaches to further improve service. This was the motivation to design Revive based on the reactive and the proactive link failure recovery schemes. The experimental results in emulated and real topologies show that Revive can improve both the TCAM usage and the failure recovery time. Thus, Revive may support applications that demand strict time requirements like data centres, cloud or carrier-grade networks. Revive adopted a k-edge connected routing topology that guarantees $k-$edge disjoint routes among every pair of switches and supports a wide range of spanning structures to meet various application demand (optimal distance, hops, or energy). These spanning structures furthermore guarantees a route-stretch factor close to 1.0, i.e., the disjoint backup route does not deviate much from the optimal value. These properties help Revive to enjoy low hop-stretch and high throughput. Revive then consolidated the memory usage among the switches to utilize the network resources efficiently and further enhanced the network throughput while meeting the application policy. This is done by exploiting the $k-$edge disjoint topology and the global network monitoring, traffic analysis, and dynamic network programming ability of the SDN architecture.

VI. Conclusions

In this paper, we have presented Revive, a hybrid approach of recovering from link failures at the data plane of SDN. It has exploited the advantages of the reactive (optimal memory uses) and proactive (optimal recovery time) recovery schemes to offer efficient memory utilization and small recovery time. Furthermore, Revive could choose any spanning structures as the routing topology to meet the application policy. Finally, Revive consolidated the TCAM usages among the switches between a source-destination pair without violating the application policy and reliability. Experimental results on emulated and real topologies revealed that Revive could be the right choice for applications like data centers or cloud networks to offer reliable services. In addition, it could provide the application policy enforcement and the network performance while efficiently utilize the network resources.

References

[1] M. Casado *et al.*, "Rethinking enterprise network control," *IEEE/ACM Transactions on Networking*, vol. 17, no. 4, pp. 1270–1283, Aug. 2009.

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

[3] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined wan," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, Aug. 2013.

[4] M. Darianian, C. Williamson, and I. Haque, "Experimental evaluation of two openflow controllers," in *T2017 IEEE 25th International Conference on Network Protocols (ICNP)*, October 2017.

[5] I. Haque and N. Abu-Ghazaleh, "Wireless software defined networking: a survey and taxonomy," *IEEE Communications Surveys and Tutorials*, 2016, (forthcoming).

[6] V. Kolar, I. T. Haque, V. P. Munishwar, and N. B. Abu-Ghazaleh, "CTCV: A protocol for coordinated transport of correlated video in smart camera networks," in *ICNP*. IEEE Computer Society, 2016, pp. 1–10.

[7] P. Fonseca and E. Mota, "A survey on fault management in software-defined networks," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 4, pp. 2284–2321, 2017.

[8] Y. Lin, H. Teng, C. Hsu, C. Liao, and Y. Lai, "Fast failover and switchover for link failures and congestion in software defined networks," in *ICC*. IEEE, 2016, pp. 1–6.

[9] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "Research challenges for traffic engineering in software defined networks," *IEEE Network*, vol. 30, no. 3, pp. 52–58, June 2016.

[10] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter, "Traffic engineering with forward fault correction," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 527–538, Aug. 2014.

[11] H. Liu *et al.*, "Traffic engineering with forward fault correction," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14, 2014, pp. 527–538.

[12] I. Haque, S. Islam, and J. Harms, "On selecting a reliable topology in wireless sensor networks," in *Proceedings of the 2015 IEEE International Conference on Communications*, ser. ICC '15, 2015.

[13] S. A. Astaneh and S. Shah-Heydari, "Optimization of SDN flow operations in multi-failure restoration scenarios," *IEEE Trans. Network and Service Management*, vol. 13, no. 3, pp. 421–432, 2016.

[14] "Mininet," http://mininet.org.

[15] "Ryu Controller," http://osrg.github.com/ryu/.

[16] "Open vSwitch," http://openvswitch.org/.

[17] O. S. Specification, "Version 1.3.0," 2013.

[18] B. Stephens, A. L. Cox, and S. Rixner, "Scalable multi-failure fast failover via forwarding table compression," in *Proceedings of the Symposium on SDN Research*, 2016, p. 9.

[19] Z. Zhu, Q. Li, M. Xu, Z. Song, and S. Xia, "A customized and cost-efficient backup scheme in software-defined networks," in *Network Protocols (ICNP), 2017 IEEE 25th International Conference on*. IEEE, 2017, pp. 1–6.

[20] A. Ghannami and C. Shao, "Efficient fast recovery mechanism in software-defined networks: multipath routing approach," in *Internet Technology and Secured Transactions (ICITST), 2016 11th International Conference for*, 2016, pp. 432–435.

[21] N. L. Van Adrichem, B. J. Van Asten, and F. A. Kuipers, "Fast recovery in software-defined networks," in *Software Defined Networks (EWSDN), 2014 Third European Workshop on*, 2014, pp. 61–66.

[22] "Bidirectional Forwarding Detection (BFD) PRotocol," https://tools.ietf.org/html/rfc5880.

[23] H. Kim, M. Schlansker, J. R. Santos, J. Tourrilhes, Y. Turner, and N. Feamster, "Coronet: Fault tolerance for software defined networks," in *Network Protocols (ICNP), 2012 20th IEEE International Conference on*, 2012, pp. 1–2.

[24] H. Liaoruo, S. Qingguo, and S. Wenjuan, "A source routing based link protection method for link failure in sdn," in *Computer and Communications (ICCC), 2016 2nd IEEE International Conference on*, 2016, pp. 2588–2594.

[25] A. Xie, X. Wang, G. Maier, and S. Lu, "Designing a disaster-resilient network with software defined networking," *arXiv preprint arXiv:1602.06686*, 2016.

[26] V. Padma and P. Yogesh, "Proactive failure recovery in openflow based software defined networks," in *Signal Processing, Communication and Networking (ICSCN), 2015 3rd International Conference on*, 2015, pp. 1–6.

[27] V. Muthumanikandan and C. Valliyammai, "Link failure recovery using shortest path fast rerouting technique in sdn," *Wireless Personal Communications*, vol. 97, no. 2, pp. 2475–2495, 2017.

[28] M. Kuźniar, P. Perešíni, N. Vasić, M. Canini, and D. Kostić, "Automatic failure recovery for software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, pp. 159–160.

[29] H. Chen and T. Benson, "The case for making tight control plane latency guarantees in sdn switches," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '17, 2017, pp. 150–156.

[30] R. MacDavid, R. Birkner, O. Rottenstreich, A. Gupta, N. Feamster, and J. Rexford, "Concise encoding of flow attributes in sdn switches," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '17, 2017, pp. 48–60.

[31] S. Zhang, Q. Zhang, A. Tizghadama, B. Park, H. Bannazadeh, R. Boutaba, and A. Leon-Garciaa, "Tcam space-efficient routing in a software defined network," *Computer Networks*, vol. 125, pp. 26–40, 2017.

[32] "Electric Lightwave," http://www.electriclightwave.com.

[33] "iPerf," https://iperf.fr/iperf-doc.php.

[34] J. Tapolcai, B. Vass, Z. Heszberger, J. Biró, D. Hay, F. A. Kuipers, and L. Rónyai, "A tractable stochastic model of correlated link failures caused by disasters," in *Proc. IEEE INFOCOM*, Honolulu, USA, Apr. 2018.