SD-FAST: A Packet Rerouting Architecture in SDN

M A Moyeen, Fangye Tang, Dipon Saha, and Israat Haque
Faculty of Computer Science, Dalhousie University
Halifax, NS, Canada
Email: {mamoyeen, fangye.tang, dipon.saha, israat}@dal.ca

***Abstract*—Communication link failure is common in any network. In Software Defined Network (SDN), protection-based recovery scheme reduces the failure recovery delay by installing alternative routes at the data plane switches. We can deploy Fast Failure Group (FFG) of OpenFlow protocol if a switch has an alternative path towards the destination; otherwise, the switch can use crankback approach to send the affected traffic towards the traversed route to find an alternative path. These existing recovery schemes force every packet to traverse a chain of matching tables even in the absence of a link failure, which impacts packet processing time and end-to-end delay. In this paper, we propose a packet rerouting architecture, called SD-FAST, that invokes recovery scheme only after facing failure and reduces both the packet processing and crankback backtracking time. We evaluate SD-FAST in Mininet, considering real and simulated traffic on real network topologies. The evaluation results confirm that SD-FAST can reduce around 73% crankback backtracking time and 64% delay compared to its counterparts.**

I. Introduction

Software-Defined Networking (SDN) [1], [2] is a new approach to designing networks, where network control logic (control plane) is separated from the network forwarding elements (data plane). The control plane translates high-level network policies to network configuration rules and installs those rules to the data plane elements using a protocol like OpenFlow [3]. SDN-based designs are widely adopted in data centers, enterprise networks, or wide area networks, where communication link failure is common. A link failure can occur as a result of damage on a network interface, data plane element, or cable. This failure, however, disrupts ongoing services [4].

The link failure recovery schemes in SDN include *restoration* [5] and *protection* [6]. In the restoration approach, a switch contacts the controller after detecting a failure to get a restoration path. In the case of the protection scheme, the controller pre-installs flow rules at each switch to enable local failure recovery. OpenFlow protocol defines Fast Failover Group (FFG) [7] to allow a switch rerouting the affected traffic without contacting the controller. The protection scheme thus reduces the communication delay between the data and control plane to bound the recovery time to 50 ms [8]. A switch can monitor the status of each port using Bidirectional Forwarding Detection (BFD) [9]. BFD exchanges periodic control messages among the connected ports and declares a link failure after missing the response of a configurable number of consecutive control packets, i.e., it provides the liveliness of a port.

In FFG-based recovery, a fast kernel data-path table maintains the packet matching rules and guides the packet to a group table to get action. However, the FFG-based solution demands at least one alternative route at each switch to locally recover from a link failure. In practice, many topologies may not have such alternative routes. In such topologies, we can deploy *crankback (CB)* approach [10]. In crankback, if a switch does not have any alternative route towards a destination, it sends back the affected traffic towards the source. In that case, every affected packet backtracks towards the source to find an alternative route, which introduces delay. Fangye *et al.* [11] suggest combining crackback with the restoration approach to reduce that backtracking delay, we name their approach *crankback with controller (CBC)*.

However, the major limitation of FFG and crankback is that every packet has to go through the group or state table. Specifically, a packet from a flow that does not experience any failure needs to be matched with a chain of tables. Matching each packet with additional state tables can incur delay, especially in the case of a long route or high load. Furthermore, in the case of crankback, a switch cannot locally terminate such backtracking as it does not have any knowledge about a remote failure. Thus, it is necessary to have a failure recovery scheme, where only the traffic facing a link failure traverses the group or state table of FFG and crankback, respectively.

Contribution: In this paper, we introduce a fast packet reroute scheme in the presence of a link failure, called *SD-FAST*, that acts only on the traffic facing failure and enables regular traffic following conventional flow tables. The proposed solution furthermore terminates crankback backtracking without controller's intervention. SD-FAST is implemented at each switch in the data plane, where its link status monitoring module uses BFD protocol to detect a link status. In the case of a link failure, SD-FAST updates the affected flow rules using the backup rules from a table maintained at the OVSDB. Note that this backup table is only accessed in the case of a link failure while regular packets follow the conventional flow tables. A switch also updates the state of the affected packets to terminate the backtrack chaining.

We evaluate the performance of SD-FAST in Mininet [12] using Open vSwitch *(OVS)* [13] and a Ryu controller [14]. We consider real network topologies USNET [7] and Darkstrand [15] where hosts exchange both the real and simulated iPerf [16] or ICMP [17] traffic. Our evaluation results indicate that SD-FAST can reduce 73% crankback backtracking delay com-

978-3-903176-24-9 © 2019 Crown

pared to CBC and 64% average end-to-end delay compared to the CB approach.

Organization: The rest of the paper is organized as follows. We compare and contrast SD-FAST to relevant research work in Section II. In the following section, we present the design and architecture of SD-FAST. Section IV depicts the evaluation setup following the discussions on the results. We conclude the paper with future research directions in Section VI.

II. Related Work

In this section, we compare and contrast existing relevant literature with SD-FAST. We categorize existing related work into FFG, CB, or source routing based techniques.

Recovery using FFG: The designs in [18], [19], [20], [21], [22], [23] deploy FFG to implement local failure recovery. Ghannami *et al.* [18] define a set of pre-computed rooted trees as the primary and backup routes, where FFG is used at each node to redirect the traffic to the backup tree. Cheng *et al.* [20] define a weighted capacity matrix to avoid congestion while redirecting the traffic to a backup route using FFG. A similar congestion avoidance scheme is proposed in [19]. Xie *et al.* use multi-stage pipeline processing for load balancing while recovering from a link failure. Revive [21] efficiently distributes TCAM usage between primary and backup routes of a flow. The solutions that rely on FFG require a network topology having alternative routes between every pair of nodes. Furthermore, regular traffic needs to travel through the two-stage table matching that incurs a delay. SD-FAST does not have any requirement on the routing topology, and fail-free regular traffic follows only the conventional flow tables.

Recovery using crankback: The work in [10], [24] uses the crankback technique to provide failure recovery when a topology does not offer an alternative route between every pair of nodes. Detour [10] uses Openstate [25] and considers link capacity while construct the backup path to avoid congestion. SPIDER [24] avoids the dependency on the controller using a stateful data plane and constructs recovery routes using crankback. The main pitfall of crankback-based approach is its packet-by-packet backtracking, which does not stop until the failed link is restored. Furthermore, packets in crankback need to traverse a multi-stage chain of tables. SD-FAST stops the crankback backtracking as soon as it finds an alternative route.

Recovery using source routing: Source routing based techniques are proposed in [26], [27], [28], where a packet carries the routing information in its header. Stephens *et al.* [27] deploy source routing with the flow rules compression to achieve both the link failure recovery and efficient TCAM usage. However, the added header bits require special packet processing and may introduce packet processing delay. SD-FAST does not add any routing information in regular packets but appends a tag in the traffic facing a link failure. It can also learn about a distant link failure to terminate the crankback backtracking.

III. Architecture and Design

In this section, we present the SD-FAST architecture and operation. The architecture is depicted in Figure 1, where different network functions are decomposed into modules. The *route planner* and other network application modules reside in the management plane and contact the controller through the northbound API. The main functional modules in the controller include *topology control*, *route configuration*, and *statistic collection*. The Ryu controller uses OpenFlow 1.3 to install flow rules in the OVS switches. Remaining functional modules of SD-FAST reside in the data plane, which include *link and packet status monitor*, *rule changer*, and *rule grabber*. In the following, we present the design and operation of different functional modules.

A. Management and control plane modules

The *topology control* module gets the network state information from the *statistic collection* module that periodically gathers the network state information from the data plane elements. Furthermore, the data plane elements send any critical change in the local topology (e.g., link failure) to the *statistic collection* module. The *topology control* module use the collected state information to construct and maintain the network topology. We use the topology construction algorithm presented in [29] to generate the routing topologies (a subgraph of the physical topology). The routes can be constructed using the shortest path tree (SPT) or minimum spanning tree (MST) depending on the applications demand.

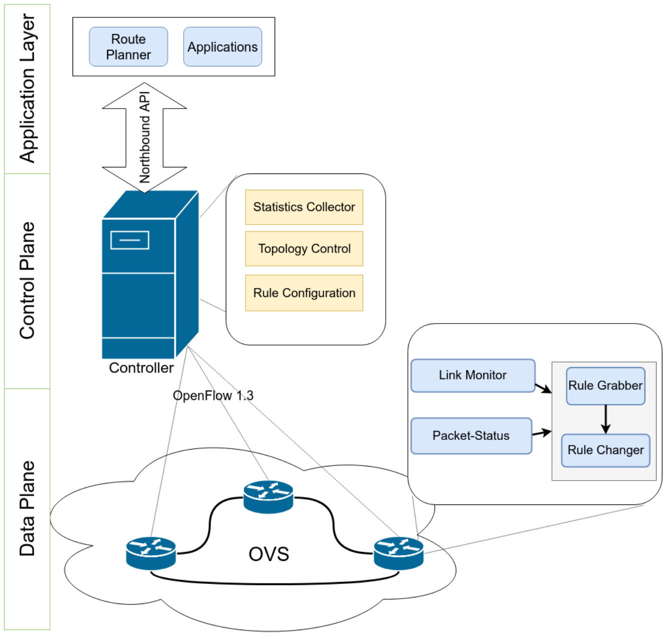


Fig. 1. The Architecture of SD-FAST.

Given a network topology, $G(V,E)$, where V and E are the set of nodes and vertices, respectively, the *route planner module* constructs two routes between every pair of nodes.

Depending on the topology, these created routes will be edge-disjoint, or they will share edges. For instance, every node in USNET topology has at least two paths to reach other nodes, which is not the case in Darkstarnd topology. The constructed routes in the former topology will be edge-disjoint, but they will share edges in the latter topology.

The *route configuration* module uses the constructed routing topology to determine the output ports for a given source-destination pair, (s,d). It starts with the primary route between (s,d) and recursively sets two output ports (primary and backup) at every node until it reaches the destination. Note that intermediate nodes between (s,d) may or may not have an alternative route to the destination depending on the given topology. In the case of an available alternative path, we can deploy FFG as well as SD-FAST; otherwise, we have to use CB, CBC, or SD-FAST.

After determining the required pair of output ports, the route configuration module installs *only* the primary route (port) in the flow table at every switch between (s,d). The backup port is kept in a *backup table* inside OVSDB. In the case of a link failure, a switch fetches this backup port to replace the primary one in the flow table. Thus, unlike FFG and CB, each packet in SD-FAST does not need to traverse a chain of tables. Furthermore, SD-FAST always uses the flow table from the kernel-space and maintains the backup table in the user-space. This decomposition helps SD-FAST achieving better processing time compared to FFG and CB.

B. Data plane modules

The *link-status monitor* module uses the BFD protocol to monitor the link status. Recall that BFD exchanges control packets at a regular interval between adjacent links and declares a link failure after missing a certain number of control packets. The *packet-status monitor* module checks the status of a packet, where a packet can be in a status of *regular*, *crankback*, or *recovered*. A packet facing no failure has the *regular* status, whereas a packet facing a failure has either *crankback* or *recovered* status. While a packet crankbacks its associated status is *crankback*, which changes to the *recovered* status as soon as the packet finds an alternative route. The recovered status is used at the destination to update the reverse-path for a given source-destination pair.

In Figure 2, PKT, PKT_{CB} and PKT_R represents regular, crankback, and recovered status, respectively. For a given source-destination pair (A,E), if the link between D and E along the primary route fails, SD-FAST changes the status of that packet, PKT, at D from ”regular” to ”crankback”. Once that packet, PKT_{CB}, reaches B, its status changes to ”recovered”.

When a switch detects a link failure, it needs to update the route and the packet status. The *rule grabber* module of SD-FAST accesses the backup table in the user-space for relevant backup routes. We use the destination IP address along with the source and destination port to find appropriate backup rules for the affected traffic. The *rule changer* module replaces the current forwarding rules with the grabbed backup ones. The affected packet then follows the new route along with the updated packet status (crankback or recovered). The operation of the *rule changer* module is presented in Algorithm 1.

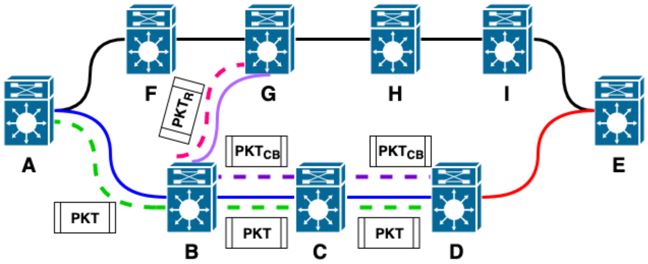


Fig. 2. Different types of packet status in SD-FAST.

```
Algorithm 1 Rule Changer Algorithm
Input:
    Selected Backup Rules: $r_b$ where, $r_b \in R$
    Destination Status: $dst$
Output:
    Rule Change Status: $s_r$
for each $rule \in r_b$ do
    if $!dst$ then
        if $in\_port(rule) != out\_port(rule)$ then
            pushRcInst($rule$)
        else
            pushCbInst($rule$)
        end if
    end if
    push $rule$ to flow table
end for
update $s_r$
```

We use Algorithm 1 to update affected flow rules with the chosen backup rules, $r_b \subseteq R$, where R is the set of backup rules. For each rule, we first check if the associated packet, P, reached the destination. If a packet reaches the destination, we update the associated reverse path towards the source. Otherwise, we first change the packet status (crankback or recovered), then update corresponding affected flow rules in the primary table using the backup ones.

We need to search both in the primary and the backup flow tables to find and update the affected flow rules with the backup rules. Suppose there are S and R rules in the primary and backup tables, respectively. A single link failure can affect multiple flows. Suppose there are f flows that are affected by a link failure. Then, the time complexity of Algorithm 1 is $\mathcal{O}(f \log R)$.

C. Local Packet Rerouting

In SD-FAST, when a fail-free packet comes to an interface, it uses the flow rules from the primary flow table. However, we deploy Algorithm 2 to locally reroute packets in the presence of a link failure. The algorithm uses the incoming packet P and link status L_s to detect a remote or local failure, respectively.

In the case of local failure (one of the links of a switch), the algorithm executes steps 4 to 6. In particular, *rule grabber* and *rule changer* of that switch update (using Algorithm 1) the affected flow rules in the primary table using the backup rules.

In the case of a remote failure, the algorithm 2 implements steps 7 to 13, where a switch first checks the packet status, then executes packet forwarding actions following Algorithm 1. Finally, the switch updates the packet status before releasing it to the output port. This status update depends on whether the packet reaches the destination or not. In the former case, the algorithm updates the reverse path of that packet. However, if the packet is still in transit, the algorithm uses the current status. If the state is "crankback," it is updated to crankback or recovered; otherwise, the status is recovered.

Algorithm 2 Packet Rerouting Algorithm

```
Input:
    Packet: $P$
    Link Status: $L_s$
Output:
    Rule Change Status: $S$
 1: $S \leftarrow 0$
 2: $dst \leftarrow 0$
 3: $IP \leftarrow Extract(P)$
 4: if $isDown(L_s)$ and $isEmpty(S[IP])$ then
 5:     $R \leftarrow RuleGrabber()$
 6:     $S[IP] \leftarrow RuleChanger(R, dst)$
 7: else if $(isCbtag(P)$ or $isRctag(P))$ and $isEmpty(S[IP])$
    then
 8:     $R \leftarrow RuleGrabber()$
 9:     if $isCbtag(P)$ then
10:         $S[IP] \leftarrow RuleChanger(R, dst)$
11:     else if $isRctag(P)$ and $isDest(P)$ then
12:         $dst \leftarrow 1$
13:         $S[IP] \leftarrow RuleChanger(R, dst)$
14:     end if
15: end if
```

IV. Evaluation Setup

This section presents the emulation environment that we use to evaluate SD-FAST. We use Mininet 2.3, Open vSwitch 2.9.1, and a Ryu controller 4.30 on a server consisting of 2.66GHz 12 core CPU, 44GB RAM, and Ubuntu 16.04.3 LTS operating system. The data plane elements, which are OVS switches, communicate to the Ryu controller using OpenFlow 1.3 protocol. The BFD control packet exchange interval is 5 ms, and a three consecutive failed response declares a link failure.

Table I summarizes different parameters that we use to evaluate SD-FAST. We choose USNET topology as it offers edge-disjoint routes among nodes. On the other hand, Darkstrand has crankback routes. We evaluate the performance of FFG, SD-FAST, CB, and CBC over these two topologies, where red-colored nodes are the source-destination pairs. We can generate a different possible combination out of these pairs. Thus, we consider sixty source-destination pairs to exchange traffic, where we randomly select twelve pairs at a time to enable their concurrent communication. We repeat the process five times to cover all sixty pairs, where each time a unique set of twelve pairs are chosen.

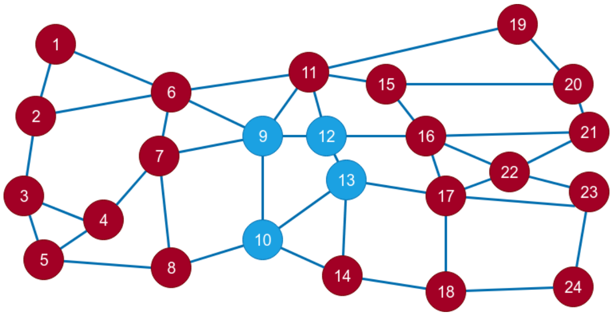


Fig. 3. Twenty four node USNET topology.

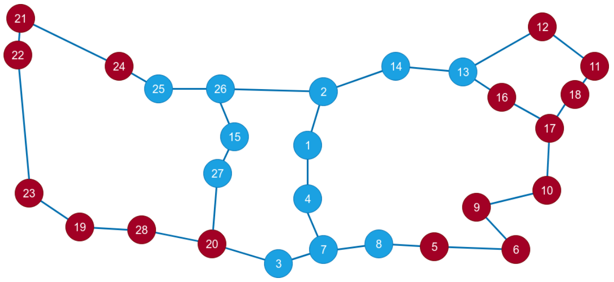


Fig. 4. Twenty eight node Darkstrand topology.

As part of the packet generation, we use both the real and iPerf (UDP) traffic. In the case of real traffic, we exchange an mp4 file among the source-destination pairs. Otherwise, we use ICMP ping and iPerf to measure the delay and throughput, respectively. In the case of real traffic, data size varies from 200MB to 350MB. On the other hand, the source-destination pairs exchange 3 MB iPerf or ICMP traffic, where each of the twelve pairs exchanges traffic for 10 minutes. Finally, we concurrently fail one link along the primary paths of the chosen twelve pairs.

TABLE I
Summary of different evaluation parameters.

Parameters	**Description**
Topology	24 node USNET (Figure 3) and 28 node Darkstrand (Figure 4)
Traffic generation	ICMP ping, iPerf based UDP, wget to transfer mp4 file.
Source-destination pairs	60 source-destination pairs, where always twelve pairs communicate concurrently.
Link failure	One link failure in each of the twelve primary paths.

We measure the average throughput, end-to-end delay, and backtracking time according to the definitions presented in Table II.

TABLE II
THE DEFINITION OF METRICS.

Name	Definitioin	Unit
End-to-End delay	The round trip time of a packet or file transfer.	ms, sec
Throughput	The rate of successfully delivered traffic.	Mbps, MBps
Backtracking time	The time difference between a link failure and crankback termination, i.e., the recovery time .	ms

V. Discussion on Results

In this section, we discuss the evaluation results of SD-FAST. We furthermore compare its performance with FFG, CBC, and CB. We categorize the results into the impact of topology, link failure, backtracking, and real traffic.

The impact of topology: Figure 5 presents the average end-to-end delay in the presence of link failures in USNET and Darkstrand topologies. Each node in USNET topology has at least one alternative route towards any other nodes, which helps a packet to recover from a link failure quickly. In the absence of backup paths at each node, a packet in Darkstrand topology follows crankback and experiences backtracking delay. Thus, all four schemes experience a significantly higher delay in Darkstrand compared to USNET topology. In particular, CB has the worst performance, while SD-FAST is the best scheme. Note that FFG requires alternative routes towards the destination to recover from a failure; thus, its performance is absent in Darkstrand.

If we want to reduce the delay over various type of topologies, SD-FAST is the right choice as it minimizes the delay compared to the other schemes. However, in the case of moderate size iPerf or ICMP traffic in USNET topology over an average route length, the improvement is not significant (Figure 5). This performance tells that SD-FAST is more effective in the topology like Darkstrand with a large number of crankback routes. In particular, SD-FAST improves around 54% and 27% average delay compared to CB and CBC, respectively. We suspect that this performance enhancement of SD-FAST is because of its recovery approach using the packet status and quick backtracking termination.

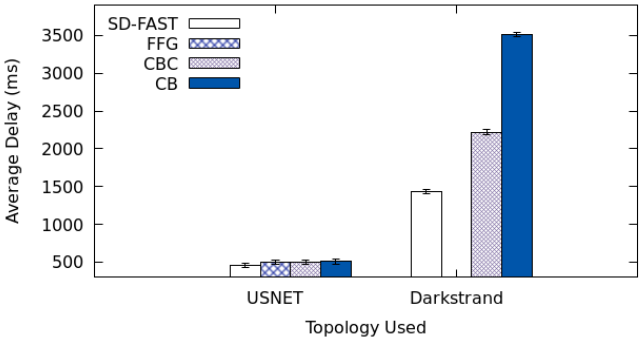


Fig. 5. The average end-to-end delay in USNET and Darkstrand topology.

We present the average throughput in Figure 6. We observe an expected performance trend similar to that of the delay. In particular, the performance difference of different schemes is more prominent in Darkstrand topology, where SD-FAST outperforms others.

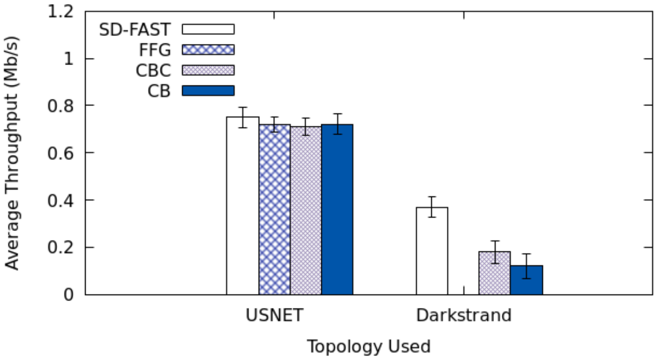


Fig. 6. The average throughput in USNET and Darkstrand topology.

The impact of link failure: We measure the impact of a link failure in Darkstrand topology as it has a large number of crankback routes, which has an impact on the four schemes. In this evaluation, we vary the percentage of the link failure, where the primary routes of the chosen pairs get disconnected, i.e., the affected traffic has to follow backup routes. However, we make sure that the topology is still connected to carry the ongoing traffic. Figure 7 and 8 present the average delay and throughput, respectively.

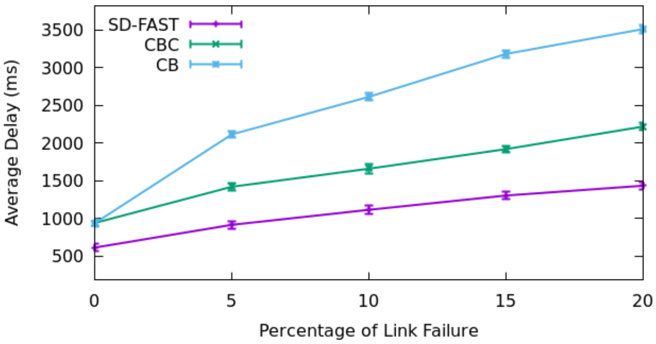


Fig. 7. The average end-to-end delay in the presence of link failure.

SD-FAST reduces around 60% and 28% average end-to-end delay while improves around 57% and 27% average throughput compared to CB and CBC, respectively. The link failure forces the traffic to follow a longer backup route and enables crankback route. Both CB and CBC suffer from the backtracking and chain of table matching delay, which is not the case in SD-FAST. The delay increases with the increasing percentage of the link failure in all three schemes, where CB has the worst performance. Note that FFG requires backup routes towards (no backtracking) the destination to recover from a failure; thus, its performance is not available for Darkstrand.

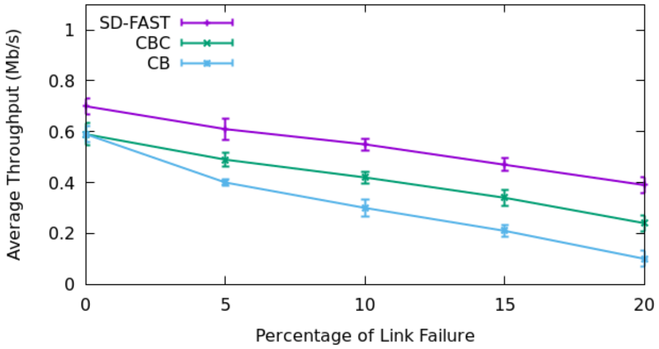


Fig. 8. The average Throughput in the presence of link failure.

The convergence of crankbacking: We consider Darkstrand topology to evaluate the backtracking convergence time. In particular, we measure how long it takes for a packet to find an alternative route after detecting a link failure so that the crankbacking process terminates. We furthermore consider 20 percentage of the link failure, which is the largest one before the topology gets disconnected. Note that the crankbacking process continuously runs at the data plane switches in the case of CB. Thus, in this set of evaluations, we compare SD-FAST with CBC. The average convergence time is presented in Figure 9.

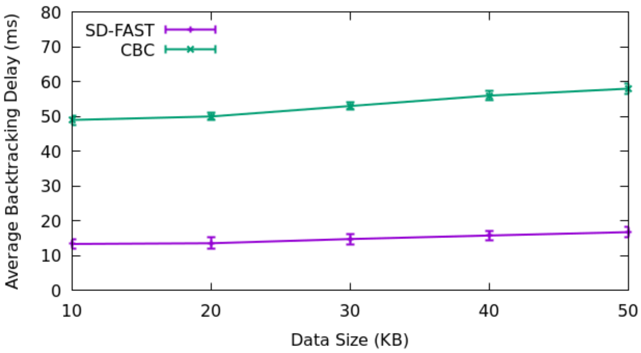
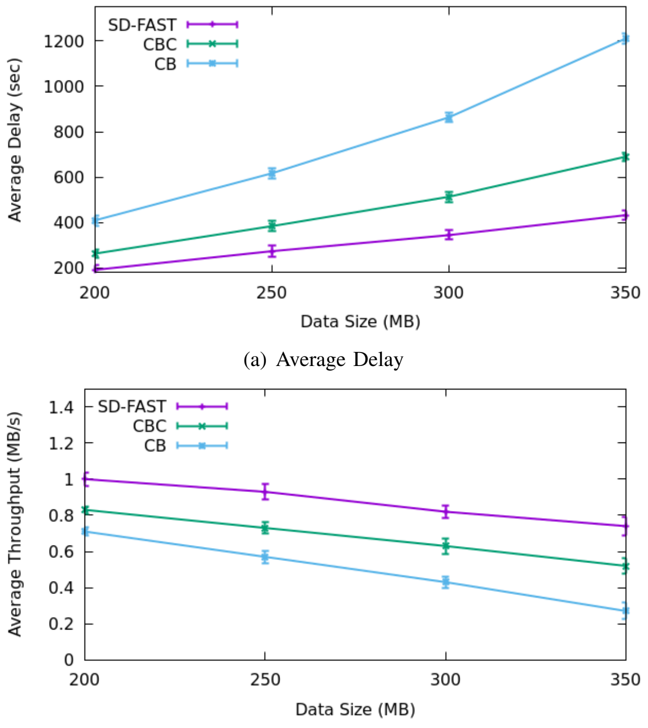


Fig. 9. The average convergence time of crankbacking in Darkstrand topology.

SD-FAST reduces 73% of the backtracking delay compared to CBC. CBC is a reactive approach, where a switch reports any link failure to the controller to update all affected routes. However, the traffic in transit between the source and failed link deploys crankback. The controller communication delay along with the chain of tables matching introduce high convergence time on CBC, which is not the case in SD-FAST.

The impact of real traffic: In the above evaluation, we observe that SD-FAST is more effective in Darkstrand than in USNET, especially over long routes. However, we suspect that the data size also has an impact on the performance of SD-FAST, CB, and CBC. In particular, SD-FAST avoids the chain of table matching, especially for the regular fail-free traffic. In this evaluation, we transfer a large size MP4 file to the chosen destinations to further illustrate the benefit of SD-FAST. The average delay and throughput of three recovery schemes are presented in Figure 10.



(a) Average Delay

(b) Average Throughput

Fig. 10. The average end-to-end delay and throughput in Darkstrand topology.

The evaluation results indicate that SD-FAST is around 64% better compared to CB in terms of the average end-to-end delay. The performance improvement of SD-FAST comes from its reduced table matching approach for the regular and failed traffic. In the case of throughput, we observe a performance trend similar to that of the delay. The throughput of CB is around 62% lower than that of SD-FAST. The performance of CBC is in between SD-FAST and CB. Both CB and CBC have a chain of table matching even for fail-free traffic, which has an impact on the performance while transferring high volume data over long routes.

VI. Conclusions

In this paper, we have introduced SD-FAST, a local on-demand packet rerouting architecture. It has removed the need for a chain of tables matching while forwarding regular and failed traffic. The evaluation results have confirmed that the chain of table processing has an impact on the performance of FFG, CB, and CBC, where SD-FAST is a winner to improve the average delay and throughput. In particular, SD-FAST has reduced 64% average end-to-end delay compared to CB and 73% average backtracking convergence delay compared to CBC. Overall, SD-FAST has significantly improved the substantial data transfer time over both the fail-free and faulty networks compared to its counterparts. As part of our future work, we plan to integrate SD-FAST modules in the OVS core and measure its performance in a real testbed.

References

[1] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 1–12, Aug. 2007.

[2] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, and S. Shenker, "Rethinking enterprise network control," *IEEE/ACM Transactions on Networking (TON)*, vol. 17, no. 4, pp. 1270–1283, August 2009.

[3] N. McKeown *et al.*, *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[4] P. Fonseca and E. Mota, "A survey on fault management in software-defined networks," *IEEE Communications Surveys & Tutorials*, 2017.

[5] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Enabling fast failure recovery in openflow networks," in *2011 8th International Workshop on the Design of Reliable Communication Networks (DRCN 2011)*. IEEE, 2011, pp. 164–171.

[6] 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*. IEEE, 2015, pp. 1–6.

[7] 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*. IEEE, 2014, pp. 61–66.

[8] K. Qiu, J. Yuan, J. Zhao, X. Wang, S. Secci, and X. Fu, "Efficient recovery path computation for fast reroute in large-scale software defined networks."

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

[10] A. Capone, C. Cascone, A. Q. Nguyen, and B. Sanso, "Detour planning for fast and reliable failure recovery in sdn with openstate," in *Design of Reliable Communication Networks (DRCN), 2015 11th International Conference on the*. IEEE, 2015, pp. 25–32.

[11] F. Tang and I. Haque, "Remon: A resilient flow monitoring framework," in *2019 Network Traffic Measurement and Analysis Conference (TMA)*. IEEE, 2019.

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

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

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

[15] "Darkstrand," http://www.topology-zoo.org/maps/Darkstrand.jpg.

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

[17] T. Chin, M. Rahouti, and K. Xiong, "End-to-end delay minimization approaches using software-defined networking," in *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*. ACM, 2017, pp. 184–189.

[18] 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*. IEEE, 2016, pp. 432–435.

[19] 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.

[20] Z. Cheng, X. Zhang, Y. Li, S. Yu, R. Lin, and L. He, "Congestion-aware local reroute for fast failure recovery in software-defined networks," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 9, no. 11, pp. 934–944, 2017.

[21] I. Haque and M. Moyeen, "Revive: A reliable software defined data plane failure recovery scheme," in *2018 14th International Conference on Network and Service Management (CNSM)*. IEEE, 2018, pp. 268–274.

[22] D. Merling, W. Braun, and M. Menth, "Efficient data plane protection for sdn," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, 2018, pp. 10–18.

[23] A. Xie, X. Wang, W. Wang, and S. Lu, "Designing a disaster-resilient network with software defined networking," in *2014 IEEE 22nd International Symposium of Quality of Service (IWQoS)*. IEEE, 2014, pp. 135–140.

[24] C. Cascone, L. Pollini, D. Sanvito, A. Capone, and B. Sanso, "Spider: Fault resilient sdn pipeline with recovery delay guarantees," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*. IEEE, 2016, pp. 296–302.

[25] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "Openstate: programming platform-independent stateful openflow applications inside the switch," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 44–51, 2014.

[26] B. Stephens, A. L. Cox, and S. Rixner, "Plinko: Building provably resilient forwarding tables," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*. ACM, 2013, p. 26.

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

[28] 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*. IEEE, 2016, pp. 2588–2594.

[29] 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.