Reliability Aware Multiple Path Installation in Software-Defined Networking

Syed Mohsan Raza 1,2, Shohreh Ahvar 3, Rashid Amin 1* and Mudassar Hussain 4

1 Department of Computer Science, University of Engineering and Technology, Taxila 47080, Pakistan; smohsanraza@gmail.com or mohsan.raza@abasyn.edu.pk
2 Department of Computing, Abasyn University, Peshawar 25000, Pakistan
3 ISEP-Institut Supérieur d’Électronique de Paris, 75006 Paris, France; shohreh.ahvar@isep.fr
4 Department of Computer Science, University of Wah, Wah Cantt 47080, Pakistan; mhtarar@gmail.com
* Correspondence: rashid4nw@gmail.com; Tel.: +92-51-9047-848

Abstract: Link failures frequently occur in communication networks, which negatively impacts network services delivery. Compared to traditional distributed networks, Software-Defined Networking (SDN) provides numerous benefits for link robustness to avoid services unavailability. To cope with link failures, the existing SDN approaches compute multiple paths and install corresponding flow rules at network switches without considering the reliability value of the primary computed path. This increases computation time, traffic overhead and end-to-end packets delay. This paper proposes a new approach called Reliability Aware Multiple Path Flow Rule (RAF) that calculates links reliability and installs minimum flow rules for multiple paths based on the reliability value of the primary path. RAF has been simulated, evaluated and compared with the existing approaches. The simulation results show that RAF performs better than the existing approaches in terms of computation overhead at the controller and reduces end-to-end packet delay and traffic overhead for flow rules installation.

Keywords: failure handling; link failure; OpenFlow; reliability; software-defined networking

1. Introduction

Software-Defined Networking (SDN) has emerged as a new network architecture that physically separates the control and application planes from the data plane of forwarding devices by implementing these at a logically centralized entity, called the SDN controller [1,2]. Link failure resiliency is considered an important benefit of SDN that provides a better Quality of Service (QoS) to the users. Link failures frequently occur in a production network, which may lead to the unavailability of network resources and service disruption. Consequently, if a link fails for a long period without appropriate actions for recovery, the network performance degrades. The existing approaches for link failure in SDN are categorized into two types, proactive and reactive, depending on their flow rule provisioning methodologies [3,4]. Restoration or the reactive failure recovery mechanism is a post-failure recovery. At any discrete time, when the controller receives the signal of a switch failure, it installs the flow rules along the alternate path. As a result, data packets start forwarding along new paths. The disadvantage of this approach is that it introduces longer delays during the process of informing the controller and subsequently computes and installs flow rules along the alternate path. Considering the topology in Figure 1 as an example, there are two paths between switches A and G. If the link between switches A and B fails for the flow between switches A and G, then switch A informs the controller and the controller computes an alternate path (A–C–D–G) as shown in Figure 1.
Figure 1. Reactive alternate flow rule installation in case of link failure.

As compared to reactive, the proactive mechanism considers the link failure resiliency in advance in the SDN production network. The controller installs flow rules along multiple paths at a switch for the communication between the source and the destination. When a switch detects a link failure, it routes the packets to the alternate path without contacting the controller. For example, in the topology in Figure 2, the controller computes two paths passing between switches A and G and installs flow rules along two paths (e.g., (A–B–G) and (A–C–D–G)). This mechanism is faster than the reactive failure recovery mechanism because there is less controller intervention during link failure [5,6]. However, there are also some disadvantages associated with this mechanism. Some of these disadvantages include longer delay of computing multiple paths, larger traffic overhead to install flow rules along multiple paths at the switches and the consumption of more memory at the flow tables of switches for storing multiple flow rules. Moreover, SDN switches have limited Ternary Content Addressable Memory (TCAM), and approximately 1500 flow rules can be accommodated or commanded by control functions [7]. TCAM is expensive in terms of cost, although it is a faster memory. Straightforward approaches of fault resiliency in OpenFlow-enabled devices (i.e., Fast Failover-based protection, individual Packet-In granular processing-based restoration) can cause TCAM overflow. However, the computing of single or least required paths contributes to the reduction of control traffic overhead and flow rule entries in TCAM.

Figure 2. Proactive flow rule installation on two communication paths.

In addition, in both reactive and proactive methods, the flow rules installation affects the QoS KPIs (i.e., overhead at controller, end-to-end packet delay, traffic overhead for flow rule installation and data plane devices memory utilization). Therefore, computing and installing fewer flow rule entries in forwarding devices can provide better QoS for users. Our proposed approach, Reliability Aware Multiple Path Flow Rule (RAF) tries to
improve the QoS for users by reactively installing the minimum number of flow rules in the forwarding switches.

In RAF, by symmetric packet exchange among the controller and switches, the controller computes the availability of links. In the case of failure occurrence, if a reactively computed primary path has a higher reliability profile, then only limited alternative flow rules are needed for expected detouring demand to mitigate the link failure. Consequently, the proposed work is a promising solution for decreasing the path computation and the reconfiguration of overhead parallel to device memory-efficient utilization. We devise two versions of RAF with slightly different objective functions: RAF calculates the primary path based on higher reliability, whereas Distance-based RAF (DRAF) computes the primary path based on the joint value of higher reliability and shorter path length. The detail of our proposed approach is given in Sections 4 and 5 and describes the simulation results. Finally, Section 6 consists of the obtained results, and Section 7 concludes the paper.

2. Related Work

This section discusses different solutions for link failure handling. Some of the solutions are reliant on controller intervention (reactive approaches), and some try to solve the single link failure recovery problems in stateful data plane fashion proactively (i.e., Group table-based OpenFlow-specified configuration utility). In context for streamlining the network robustness, the latest research also includes the software hybrid strategies to tackle failures in various network’s infrastructure.

Thorat et al. [8] aggregated the disrupted flow rules in the case of failure using the VLAN tagging and redirecting the tagged flows. The authors proposed new failure recovery techniques called Controller Independent Proactive (CIP), and Controller Dependent Proactive (CDP), which use the fast failover (FF) and indirect group tables, respectively. Lim and ICoD in [9] presented the VLAN tagging, FF and indirect group similar to the CIP and CDP. CFR in [10] is another research work that proposed the recovery mechanism based on the same principle of a VLAN tagged flow rule aggregation and local recovery supported by FF. In the second phase of the CFR, it deals with resource utilization after traffic detouring.

PREF-CP [11] addressed the problem of control channel and data plane device link failure using multiple controllers. This framework includes monitoring and reassignment modules. These modules jointly monitor the health of the controller in terms of routing requirement load. PREF-CP handles the failure among the controller and switches by tunneling the disrupted switch with a different active controller. The proposed framework leveraged a genetic algorithm that considers the controller load distribution and failure probability in a multi-controller problem space.

Comparative to the single centralized controller architecture, FT-SDN in [12] proposed using multiple open-source heterogeneous controllers in a network. This methodology indulges the multiple controller intervention in the case of primary controller link failure with a switch and exchanging the controller states in order to pave the synchronization among controllers. However, enabling the multi-controller is suitable for the large-scale network only. Since the secondary controllers on the fly management can cause the synchronization and cost elevation challenges in small softwarized networks. In this regard, in the related work, there are controller placement algorithms and assembly models to support the East–West interfacing of controllers (horizontal, vertical and hybrid models) [13–15]. These models support the homogeneous and heterogeneous control function entities with objectives of scalability and distributed computing.

Malik et al. [16] proposed the rapid failure recovery for large and medium end-to-end path failure handling in the case of contingent recovery requirements. It explored a novel approach to consider the failed path partially by partitioning the network graph logically in various communities based on similarities of the network’s nodes. When a path is failed, the recovery approach only considers the link failure solution in a specific community, contrary to complete alternative path computation or considering the rest of the path. As a
result, it decreases the operational cost of reconfiguration such that it is least demanded. On the other hand, it also elevates the computation cost of fault localization and detouring in the case of lengthy path partition. The same author proposed a smart routing, which is a proactive fault handling of SDN [17]. The SDN controller handles the flow disruption by foreseeing the failure events in smart routing. As one of the rare works based on reliability, the author in another work [18] tried to find the most reliable path for SDN failure recovery. The idea here was that the disjoint path is not always the perfect choice as a backup because it will require totally new flow entries for all nodes that form a path. Instead, they try to select the most reliable path as the backup. However, the reliable path here refers to the one containing the highest number of shared edges among all paths between a source and destination. This definition is different from the one used for RAF in this paper. In addition, the authors select only one backup path.

Destination-based tunneling mechanism (Repair Path Refinement with Destination-based Tunneling (RPR-DT)) proposed in SDN Candidate Selection (SCS) [19] introduced the shortest repair reactive path in the case of link failure. SCS decreased the number of required SDN switches for detouring (i.e., repair path) the disrupted packets in IP/SDN hybrid infrastructure. Exalting the shortest path from the permutation set of repaired paths near a failure can avoid the bandwidth underutilization. Further, it is also capable of decreasing the end-to-end delay experienced by rerouted traffic. SafeGuard [20] is a failure recovery system for SD-WAN and considers the bandwidth and the switch memory utilization. SafeGaurd formulated the failure recovery as a Mixed-integer linear Programming optimization problem. Nevertheless, SafeGuard, similar to various recovery techniques, used the fast failover group, but it also considered the alternative link and intermediate switches capability before rerouting the failure impacted flow rules. The authors claimed the efficient utilization of switch memory and congestion control in the case of failure by leveraging the SafeGuard-distributed weights, which correspond to alternative path capacity-aware recovery. In another study [21], a Multipath Resilient Routing Scheme for SDN-Enabled Smart Cities Networks (MPResiSDN) was proposed. The authors tried to use SDN to improve network resilience to external natural challenges, such as storms. They compared the algorithm with the Spanning Tree method and showed increasing the number of installed paths increases overhead. However, they have not yet investigated the best ways to determine the optimal value of the number of paths.

SPIDER [22] is a pipelined packet processing mechanism that applied the stateful design for the recovery mechanism. It is mainly composed of OpenState, in which state tables are used, and the flow tables are proceeded by the state tables. Failure detection is investigated using bidirectional heartbeat packets that determine the port aliveness status. The behavioral model of SPIDER is in the form of a finite state machine that acts as per-flow processing. Flow Table Compression in [23] utilized the OpenFlow switch memory efficiently. It addressed the problems of data center networks where the link reliability, multipath availability and immediate fast recovery are highly demanded. To ensure these demands, when a switch has multiple paths along with a primary path in its memory, this mechanism helps to increase the inefficient usage of TCAM that is a crucial resource. However, the proposed approach only considered the failure localization and reduction of delay time in flow deployment regardless of the alternate path installation or associated techniques.

Lin et al. [24] proposed a switchover mechanism to resolve the link failure problem in SDN. The authors identified the problem of considering the fast failover group table for proactive recovery. When multiple flow rules pass through a common output port of any group table action bucket, the port status is set to be congested. To address this problem, a switchover mechanism is used. Switchover swaps the action buckets and configures the less congested status port as alive output Port dynamically. Although it is an efficient technique that reveals the problem of using fast failover as an ultimate solution, the multiple ports congestion gauging overhead is not discussed in this work. CORONET [25] supports the fast failure recovery in SDN. It has a mechanism of VLAN-based communication and
various decisions at the switch level for less controller intervention. OpenFlow applications in this framework are used for computing the logical communication paths only, in contrast to various softwarized recovery solutions. CORONET uses multiple alternative paths for rerouting the configuration regardless of the calibration of the primary path health.

Self-Healing Protocol (SHP) [26] leveraged the autonomic switch mechanism in order to have the least controller intervention, lowering the global recovery time and various routing decisions in the data plane layer. Moreover, this protocol needs the supported SHP agents as collaborators of forwarding devices and extra packets for failure handling.

Panev et al. [27] incarcerated the OpenFlow-supported FF and SPIDER functionalities in 5G infrastructure. This research mainly considered the recovery latency, which is a primary concern in future 5G networks. Although the amalgamated probing provisioning, disrupted traffic tagging and fast failover mechanisms tend to be valuable directions in this work, their applicability in realistic 5G resources needs to be investigated in real-network resources. The author also highlighted the need for an update in OpenFlow switches to make them capable of the encapsulation of GPRS tunneling protocol packets.

Similarly, a prior research work proposed [28] enabling the softwarization in Evolved node B (eNB), which is an interface for user equipment (UE) in 5G. The proposed architecture deploys the OF packet and serves gateway switches in the data plane linked with the central OpenFlow controller in the out-of-band mode. The controller has reactive recovery in the case of a serving gateway switch failure and installs the new route based on the delayed sensitivity status of the user equipment demand.

Amaru [29] is a protocol that works for failure handling. It incarcerates the multi objectives (i.e., fast bootstrapping, topological scalability) in in-band SDN control. In fact, Amaru extends the Breadth-First Search algorithm functions to compute multiple alternative paths for protected resiliency support. A protocol for fast control channel recovery was proposed in [30]. It divides the communication channel in in-band SDN control communication. Moreover, it decreases the controller intervention in topological and physical interface properties elicitation. This protocol works to handle multiple simultaneous failures in a short time. Sharma et al. [31] resolved this problem by disabling the VM interface contingently, corresponding to the OpenFlow switch, when a port failure notification was received at the controller. Afterward, the controller computes a new failure-free route for communication among VMs.

From the above-presented information, summarized in Table 1, and the rest of the considered failure resiliency approaches in SDN [32–37], it can be safely concluded that none of these traffic detouring strategies hinge on the reliability status of the primary computed communication path. We have observed the local recovery mechanism of switches (e.g., indirect group and fast failover) as an ultimate inclination of the latest proactive studies. These are valid methodologies for the abrupt fast failure recovery, but these can cause further challenges, where the infrastructure budget is a constraint along with the QoS and Quality of Experience (QoE) constraints. More specifically, the fast failover group in SDN-forwarding devices is subject to optional support according to the OpenFlow specification. Secondly, failover alternative reaches could be out of service prior to the active path [38]. It tends to the less interoperability of the discussed methods in forthcoming traffic detouring demands. Although research work is available that manual fixing of failure handling based on alternate path reliability solutions, the reliability aware communication in SDN is still a challenge for its adaptation. Shared Risk Link Groups (SRLGs) help to express the relationships of complex failures. The research work [39] addressed the problem of various SRLG failure scenarios. Along with SRLGs constraints, this paper is the first attempt to provide the protection mechanisms against failures in the SDN/NFV environment. The proposed method uses the Column Generation and Bender Decomposition methods for its major objectives of saving the network against SRLG failure. Kiadehi et al. [40] also employed the SRLG for the computation of non-overlapping path redundancy to address the problem of the overlapping primary and backup paths. According to the author, the SRLG-based proposed method is more efficient
for enhancing the fault tolerance in IoT environments as compared to the Dijkstra and Disjoint Path (DP) algorithm. Similarly, EPSUTE in [41] uses SRLG for two disjoint path installations (working and backup) among the source and destination.

Table 1. Research works focusing the failure resiliency in SDN.

| Reference | Procedure | Network Infrastructure | Recovery Approach | Control Channel Connectivity | Number of Controllers |
|-----------|-----------|------------------------|------------------|------------------------------|-----------------------|
| CIP, CDP [6] | FF, indirect group, VLAN-based Flow Aggregation | OpenFlow Networks | proactive | out-of-band | single |
| LIm, ICoD [9] | FF, indirect group, VLAN-based aggregation | Internet of Things | proactive, reactive | out-of-band | single |
| CFR [10] | FF, VLAN-based flow rule aggregation | OpenFlow Networks | proactive, reactive | out-of-band | single |
| [42] | Probing, in-band control tree | OpenFlow Network | reactive | out-of-band, in-band | single |
| [43] | Probing, OpenFlow group utility | SDN-enabled data centers | reactive | in-band | single |
| AFRO [44] | Control Channel Event Recording, Shadow Controller | OpenFlow Networks | reactive | in-band | single |
| PREF-CP [11] | Controller load aware genetic algorithm | OpenFlow Networks | proactive | out-of-band | multiple |
| FT-SDN [12] | State Replication Method, Network Information Base, control channel synchronization | OpenFlow Networks | reactive | out-of-band | multiple |
| [16] | Network Node, communication path community segmentation | Large OpenFlow Network Topologies | reactive | out-of-band | single |
| RPR-DT, RPR-LB [19] | SDN candidate selection | hybrid SDN network | proactive | out-of-band | single |
| SafeGuard [20] | MILP | SD-WAN, FF | proactive | out-of-band | single |
| SPIDER [22] | Open-State, FSM | SDN | proactive | topology independent | single |
| MpressoSDN [21] | Open-State, FSM | SDN | proactive | topology independent | single |
| Smart Routing [17] | Fault Tolerance Algorithms, prediction | SDN | proactive | out-of-band | single |
| Fast failover and switchover [24] | FF, Port monitoring | SDN | proactive, reactive | out-of-band | single |
| CORONET [25] | VLAN growing algorithm | SDN | Proactive | out-of-band | single |
| [28] | Of supported state modification methods | SDN enabled 5G network infrastructure | reactive | out-of-band | single |
| DPQoAP [45] | Dynamic Adaptive Streaming over HTTP (DASH), FF | SDN | proactive, reactive | out-of-band | single |
| [46] | IP tunneling protocol | hybrid SDN | reactive | out-of-band | single |
| Amaru [29] | Recursive BFS, Mac Labeling | hybrid SDN | proactive | in-band | single |
| [30] | communication channel division, logical ring based failure identification | SDN | proactive | in-band | single |
| [47] | control and data traffic queuing, Bidirectional Forwarding Detection (BFD) | SDN | proactive, reactive | in-band | single |

3. System Model

Assume an SDN-enabled infrastructure where we have a single controller C and SDN switches supporting OpenFlow. There is an out-of-band communication model in this system in which each switch is directly connected with the controller. We have weighted links between the forwarding devices, and our system assumes a stable link always exists between the controller and the forwarding devices.

4. Proposed Approach

In this section, we present a reactive Reliability Aware Flow Rule Installation mechanism (RAF), which calculates the primary path based on higher reliability. Then, we propose a variation of RAF, Distance-based RAF (DRAF), which computes the primary path based on the joint value of higher reliability and shorter distance. RAF includes three phases, the Bootstrapping Process, graph composition process and path computation/installation phase.
4.1. Phase A: Bootstrapping Process

When an OpenFlow channel is established between the controller and switch, sym-
metric packets, such as Hello, Echo request and Echo response, are exchanged among the
controller and connected switches. The controller initiates a Feature-Request and mul-
tiple other asynchronous control requests for data plane global awareness in a periodic
stream [48]. More specifically, our mechanism uses switch states inspection, state modifica-
tion, interface statistics, interface aliveness monitoring, flow rule statistics and capabilities.
Broadly, the process is presented in the form of publisher-subscriber software applications,
which are event-driven sources for network control functions. Further, by caching these
responses of the data plane in hash-able structures along with their introduction time, the
controller maintains the network-wide view dynamically and periodically.

4.2. Phase B: Graph Composition Process

Controller C transforms the output of phase A into a weighted undirected graph
\( G = (V, D, L, L_{\text{attrib}}) \), where \( V \) is a set of \( \text{end}_\text{-user} \), \( D \) is a set of switches, \( L \) is the set of links
between them (i.e., \( L_{\text{switches}} \), \( L_{\text{end}_\text{-user}} \) and \( L_{\text{attrib}} \) shows the links’ attributes (e.g., Reliability,
Timestamp). The controller periodically updates graph connectivity in the response of
the end-user and device discovery events. Reliability inquired by the controller reflects
the stability of the link among forwarding devices and \( \text{end}_\text{user} \) connectivity with devices.

Algorithm 1: GC (C)

| Input: C |
| --- |
| /*C is a controller*/ |
| Initialize undirected Graph = \( G = (V, D, L, L_{\text{attrib}}) \), where \( V \) is a set of \( \text{end}_\text{-user} \) |
| D is a set of switches, \( L \) is the set of links between them (i.e., \( L_{\text{switches}} \) |
| \( L_{\text{end}_\text{-user}} \) and \( L_{\text{attrib}} \) shows the links’ attributes*/ |
| Procedure Activate (C(event_publisher, event_subscriber)) |
| Procedure Data_Plane (link_event, Publisher): |
| C_handler (link_event, Subscriber) |
| return devices_map \((D, L_{\text{switches}}, L_{\text{attrib}})\) |
| Procedure Data_Plane (end_user_event, Publisher): |
| C_handler (end_user_event , Subscriber) |
| return end_user_map \((V, L_{\text{end}_\text{-user}}, L_{\text{attrib}})\) |
| UPDATE \( G = (V, D, L, L_{\text{attrib}}) \) |

4.3. Phase C: Path Computation and Flow Rule Installation Process

The proposed approach has a reactive flow rule installation mechanism.
At the start of the network, the flow forwarding table did not have the flow rule
entries corresponding to the network reachability. When a data packet, \( P \), from a connected
end-user arrives at the switch, \( D \), the switch looks for the already installed corresponding
matching flow rule entry in its flow forwarding table, as Algorithm 2 shows. If there exists
a matching flow rule entry, \( R \), for the data packet, \( P \), then the switch forwards the data
packet according to the action specified in the flow table entry. Otherwise, the switch will
trigger a request in the form of the PacketIn at the controller interface to compute the flow
rule entry. To process the PacketIn request and flow rule computation, the controller will
check the ACL \( A \) to investigate whether the received packet should be allowed or denied
as per network policy. The ACL specified at a controller is a network administrator scripted
application. If the flow is denied, then the controller commands the switch, \( D \), interface for
a packet drop decision for a certain or unlimited time.

Otherwise, the controller will calculate the most reliable and alternative paths to install
after running the reliable computation process.
Algorithm 2: CP(P, R, A, D)

**Input:** P, R, A

/*P is an arrival packet*/
/*R is flow rule entry*/
/*D is the switch receiving P*/
/*A is ACL table*/

if \( P \in R \) then

Permit or Deny

else

if \( \exists Policy \in A \rightarrow (P_{SRC}, P_{DST}) \in Policy(SRC, DST) \) then

Compute \( S_{rel} \) and \( S_{ALT} \)

OpenFlow Command \([Policy \rightarrow D]\)  /*returns \( S_{rel} \) and \( S_{ALT} \)*/

4.3.1. Reliability Computation Process

The controller uses probe packets for the inspection of link failure frequency during a time slot (10 s) and computes the link reliability periodically based on Equation (1). Frequent failures cause lower reliability percentages and lead to more flow rule installation demand for the controller. In Equation (2) for a link, \( L_i \), we show the failure frequency, \( \varphi_{L_i} \), between 0 and 1. In the case that the controller accessed the lower reliability (e.g., 0.3), then it denotes the higher cumulative values (70%) of \( \varphi_{L_i} \). The proposed method did not consider only the single alive time instant, but it considers the summation of the alive instances and failures to find the reliability value of the link, while the \( \Gamma_{L_i} \) presents the link aliveness ratio in stipulated \( \lambda \) seconds (here \( \lambda \) is 10 s for each iterative experiment).

\[
\varphi_{L_i} = (1 - \frac{\text{receivedmsg}}{\text{totalmsg}}) \quad (1)
\]

It is worth mentioning that one link may be active at one moment, but it has not been active in several slots of time. This formula helps to judge the link’s reliability based on the information history for a link.

\[
\Gamma_{L_i} = (1 - \varphi_{L_i}) \quad (2)
\]

In addition, one should note that in RAF, the reliability will be calculated only for the links on the paths between \( P_{SRC} \) and \( P_{DST} \), as shown by the \( \Omega \) set. \( \Omega = \{\Omega_z | \Omega_z = \{L_1, ... L_n\}\} \). Then, the reliability of each path \( \Omega_z \) is calculated as the summation of its links reliability, as shown in Equation (3).

\[
\Gamma_{\Omega_k} = \sum_{k \in \Omega_k} \Gamma_k \text{ where } \Omega_z \in \Omega \quad (3)
\]

4.3.2. RAF Path Computation

In RAF, we select the path with the highest \( \Gamma \) (i.e., most reliable path) as the primary path, shown with \( S_{rel/m} \) and follow either of the cases in Equation (4) for computing and installing another alternate path. We have used the multi-dimensional hash-able objects to store the attribute of the link. Although there is a choice of directly using the multidimensional arrays (matrix) for storing and accessing these attributes, it elevates the accessing cost (i.e., time) [49].
Case I: If the reliability of the primary path is more than 90%, RAF does not compute and install any other alternate path. \( S_{ALT} \) presents the alternate path for flow rule installation to mitigate the link failure recovery.

Case II: If the reliability value of the primary path is between 80% and 90%, the controller installs flow rules along two alternate paths.

Case III: If the reliability value of the primary path is between 70% and 80%, the controller installs flow rules along three alternate paths.

Case IV: If the reliability value of the primary path is between 60% and 70%, the controller installs flow rules along four alternate paths.

Case V: If the reliability value of the primary path is between 50% and 60%, the controller installs flow rules along five alternate paths.

Case VI: If the reliability value of the primary path is between 0% and 50%, the controller will install flow rules along all available paths \( \Omega \). \( \Omega \) can be set by the network administrator.

4.3.3. Distance-Based RAF (DRAF) Path Computation

DRAF computes the primary path based on the joint value of higher reliability and shorter path length. If more than one path is approximately close to each other in terms of reliability, the controller will choose the path with a lower number of intermediate devices.

4.3.4. Flow Installation Process

Flow rules in forwarding devices can be installed by using the Flow_Mod command of OpenFlow specifications. OFPPC_ADD is a specific OpenFlow command used for flow rule installation. The match fields specified in the flow rule adding command are first compared for any action or decision for a data packet received at switch. Matching objects include Dl_type (opcode of IPV4, ARP, etc.), match.nw_proto (application layer protocol), match.nw_src (source IP address) and match.nw_dst (destination IP address). The controller also specifies the priority and timeout values (i.e., hard and idle timeout) of a flow rule entry. Flow rules with higher priority are followed for communication, which is the deliberate intention of the SDN programmer/administrator.

5. Simulations Setup

To evaluate the performance of the proposed approach, we have used the POX controller (2.2 eel version), Mininet Emulator and OpenFlow API. In addition, we generated fake link failures.

5.1. Mininet Emulator

Mininet emulator has the benefit of interfacing for multiple SDN control software. Mininet is easy to use by offering easy creation of SDN elements, customizing, sharing and testing SDN networks. The underlying simulated hardware includes switches, hosts, links and controllers. Further, it provides a separate virtual environment for each host for executing various host granular applications. The experimentalizing of our proposed approach includes mid-level and low-level API support of Mininet for developing the customized virtual network topologies [50].
5.2. POX Controller

POX controller provides a framework for communicating with stateless switches using OpenFlow protocol. Developers can use POX to create an SDN controller via the python language. It is a popular controller for academic research in SDN and network management applications. POX support for the Mininet interface and its event-driven model enable us for RAF and DRAF analyses easily [51]. Furthermore, POX supports OpenFlow switch specification 1.0 and 1.3 versions.

5.3. OpenFlow API

OpenFlow is a data plane device management API that is used for testing our approach. OpenFlow emerged in 2008, and its evaluation continues to switch customized management [52]. We have used its 1.0, 1.3 OpenFlow specification versions, the python programming language for management and data plane scripting because it is compatible with the POX controller. The rest of the used distributions, libraries and API of python includes Networks (python graph composition), Random (accessing iterable python object randomly), netaddr (Network Address Translation), Python low-level programming interface (TCP, UDP interfaces), multi-threading and process atomicity (Daemon Threads, locks, Semaphores).

5.4. Link Failure Process

The proposed method has the functionality to make the unavailability of the path for evaluation. Based on our knowledge, the failure pattern dataset of a data center or any other organization that has adopted the OpenFlow network is not available across the Internet. Link failure is configured at the switch egress or ingress Ethernet interface by OpenFlow command at the controller. An aliveness time $\lambda$ is initialized for each link. Failure procedure is activated randomly during the aliveness time of a link, which gradually approaches a certain threshold (here 10 s total single experiment iteration time). The controller receives information about the failure using asynchronous communication with the switch and governs the decision of the flow rule installation along an alternate path instantaneously. Moreover, our failure process binds the incoming traffic with a logical port instead of setting a physical Ethernet interface down.

5.5. Network Topology

A network topology consisting of 25 virtual end hosts and 9 OVS interconnected switches is used to test RAF performance. A single host in this network topology generates 10,000 UDP data packets with an average length of 512 bytes. Subsequent packet delay controls the data rates of socket interfaces. The results are collected using five host-generated data packets because of the machine-limited resources and a lot of production packets per thread.

6. Experimental Results

This section compares the RAF and DRAF performances with existing approaches. By existing approaches, we mean where RAF or DRAF are not used and mechanisms are searching for $\Omega$ alternative paths (e.g., [36] and SPRM [53]). We have denoted all alternative paths with the symbol $\Omega$, which is explained in Section 4.3.1. The higher $\Omega$ path installation elevates the complexity and control functioning time comparative to the RAF and DRAF approaches. While $\Omega$ is set with the administrator, here in these experiments, $\Omega$ is equal to the number of all possible alternative paths, as the network was small. During the experiments, the range of $\Omega$ changed from 1 to a maximum value of 5. We show "finding all alternative paths method" in our experiment with $\Omega$ equals 5 (its maximum).

Figure 3 presents the controller overhead. The average end-to-end delay of all path-based communication is higher than that of RAF-based communication. This end-to-end latency is incurred due to the higher controller intervention and straightforward control
channel involvement. Figure 4 elaborates RAF delay computation efficiency thanks to streamlined communication and less controller involvement.

![Graph](image1.png)

**Figure 3.** Controller overhead for path computation.

![Graph](image2.png)

**Figure 4.** Comparative average end-to-end delay of existing and reliable path communication.

RAF minimizes the installation of the memory frequent flow rule entries. When an Open switch receives frequent flow entries for installation, its memory becomes a bottleneck. Figure 5 shows a relatively smaller number of flow-forwarding rules for the installation of a switch TCAM using RAF compared to the existing approaches. However, we have observed that the RAF and DRAF present a close number of flow rule entries in this perspective. The length of the path (i.e., a variable number of intermediate forwarding devices in a path) is a factor that affects the possible number of control transmissions.
Figure 5. OpenFlow-enabled switch memory.

Figure 6 depicts the number of transmissions required for the configuration of data plane devices. It is observed that the existing approaches require the highest control traffic or commands, which encompasses the flow rule entries.

Figure 6. Control packet average transmissions.

However, DRAF has the lowest number of transmissions in this case. It is also observed that high engagement for the control channel causes failure in the case of abundant flow rules installation.

Figure 7 illustrates the observation of a flow entry encapsulated packet drop at the switch due to the memory constraint of OpenFlow-enabled forwarding devices. The proposed approach is suitable for memory-restricted data plane devices.
Observations regarding the average delay of flow rules installation in Figure 8 conclude that as the number of flow rules entries increases, the time of their installation also increases proportionally. Handling multiple applications of the controller results in the prolonged delay for device configurations. RAF and DRAF reduce the delay and synchronize the immediate configurations.

All or selected path installation procedures in the existing approaches install the useless entries in the switch memory. Figure 9 presents the experimental results by accessing the packets and number of bytes entertained by the flow rule entries in switches, and it has yielded that RAF and DRAF are much more efficient for the said situation. The proposed approach is more efficient compared to the existing approaches, as it installs only useful entries in the switches (i.e., flow rule entries with continuous packet processing counter updates).
7. Conclusions

SDN is an emerging computer network architecture that decouples both control and application planes. The control centralization has many advantages, such as easy network control and management. As link failures occur frequently in a network, to deal with this problem, the existing approaches install flow rules along multiple paths in the network without considering the reliability value of the primary path. Computing all paths causes more computation overhead at the controller and larger traffic overhead in the network. Moreover, the memory requirement in the switches for the installation of flow rules also increases. To address these problems, our proposed approach, RAF, considers the reliability level of the primary path and installs flow rules along with the number of alternate paths according to the reliability value of the primary path. More specifically, if the primary path has a higher reliability, then there is no need to install flow rules along multiple paths. The simulation results show that our proposed approach has better results by decreasing the computation load at the controller, average end-to-end delay and the traffic overhead for installing flow rules along multiple paths at the data plane. As future work, we would like to extend our proposed work by computing the link reliability using deep learning algorithms.

Author Contributions: Conceptualization and methodology: S.A. and S.M.R.; Related work: S.M.R., S.A. and M.H.; Proposed Model: S.A. and S.M.R.; Problem formulation and validation: S.A. and S.M.R.; Simulations: S.M.R.; English review and proofreading: S.A. and M.H.; review and supervision: S.A., R.A. and M.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kim, H.; Feamster, N. Improving network management with software defined networking. IEEE Commun. Mag. 2013, 51, 114–119. [CrossRef]
2. Chica, J.C.C.; Imbachi, J.C.; Botero, J.F. Security in SDN: A comprehensive survey. J. Netw. Comput. Appl. 2020, 159, 102595. [CrossRef]
3. Sambo, N. Locally automated restoration in SDN disaggregated networks. IEEE/SA J. Opt. Commun. Netw. 2020, 12, C23–C30. [CrossRef]
4. Mas–Machuca, C.; Musumeci, F.; Vizarreta, P.; Pezaros, D.; Jouët, S.; Tornatore, M.; Hmaity, A.; Liyanage, M.; Gurtov, A.; Braeken, A. Reliable Control and Data Planes for Softwarized Networks. In Guide to Disaster-Resilient Communication Networks; Springer: Cham, Switzerland, 2020; pp. 243–270.
5. Scott, C.; Wundsam, A.; Raghavan, B.; Panda, A.; Or, A.; Lai, J.; Huang, E.; Liu, Z.; El-Hassany, A.; Whitlock, S.; et al. Troubleshooting blackbox SDN control software with minimal causal sequences. In Proceedings of the 2014 ACM Conference on SIGCOMM, Chicago, IL, USA, 18 August 2014.

6. Duliritsi, Z.; Rzym, G.; Cholda, P. MPLS-based reduction of flow table entries in SDN switches supporting multipath transmission. Comput. Commun. 2020, 151, 365–385. [CrossRef]

7. Qiu, K.; Yuan, J.; Zhao, J.; Wang, X.; Secci, S.; Fu, X. Fastrule: Efficient flow entry updates for tcam-based openflow switches. IEEE J. Sel. Areas Commun. 2019, 37, 484–498. [CrossRef]

8. Thorat, P.; Jeon, S.; Choo, H. Enhanced local detouring mechanisms for rapid and lightweight failure recovery in OpenFlow networks. Comput. Commun. 2017, 108, 78–93. [CrossRef]

9. Thorat, P.; Singh, S.; Bhat, A.; Narasimhan, V.L.; Jain, G. SDN-Enabled IoT: Ensuring Reliability in IoT Networks Through Software Defined Networks. In Towards Cognitive IoT Networks; Springer: Cham, Switzerland, 2020; pp. 33–53.

10. Wang, L.; Yao, L.; Xu, Z.; Wu, G.; Obaidat, M.S. CFR: A cooperative link failure recovery scheme in software-defined networks. Int. J. Commun. Syst. 2018, 31, e3560. [CrossRef]

11. Güner, S.; Gür, G.; Alagöz, F. Proactive controller assignment schemes in SDN for fast recovery. In Proceedings of the 2020 International Conference on Information Networking (ICOIN), Barcelona, Spain, 7–10 January 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 136–141.

12. Das, R.K.; Pohrmann, F.H.; Maji, A.K.; Saha, G. FT-SDN: A fault-tolerant distributed architecture for software defined network. Wirel. Pers. Commun. 2020, 114, 1045–1066. [CrossRef]

13. Almadani, B.; Beg, A.; Mahmoud, A. Dsf: A distributed sdn control planeframework for the east/west interface. IEEE Access 2021, 9, 26735–26754. [CrossRef]

14. Li, B.; Deng, X.; Deng, Y. Mobile-edge computing-based delay minimiza-tion controller placement in sdn-iov. Comput. Netw. 2021, 193, 108049. [CrossRef]

15. Pontes, D.F.T.; Caetano, M.F.; Filho, G.P.R.; Granville, L.Z.; Marotta, M.A. On the transition of legacy networks to sdn-an analysish on the impact of deployment time, number, and location of controllers. In Proceedings of the 2021 IFIP/IEEE International Symposium on Integrated NetworkManagement (IM), Bordeaux, France, 17–21 May 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 367–375.

16. Malik, A.; de Fréin, R.; Aziz, B. Rapid restoration techniques for software-defined networks. Appl. Sci. 2020, 10, 3411. [CrossRef]

17. Malik, A.; Aziz, B.; Adda, M.; Ke, C.H. Smart routing: Towards proactive fault handling of software-defined networks. Comput. Netw. 2020, 170, 107104. [CrossRef]

18. Malik, A.; Aziz, B.; Bader-El-Den, M. Finding most reliable paths for software defined networks. In Proceedings of the 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC), Valencia, Spain, 26–30 June 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1309–1314.

19. Yang, Z.; Yeung, K.L. Sdn candidate selection in hybrid ip/sdn networks for single link failure protection. IEEE/ACM Trans. Netw. 2020, 28, 312–321. [CrossRef]

20. Shojaee, M.; Neves, M.; Haque, I. SafeGuard: Congestion and Memory-aware Failure Recovery in SD-WAN. In Proceedings of the 2020 16th International Symposium on Network and Service Management (CNSM), Izmir, Turkey, 2–6 November 2020; pp. 1–7.

21. Aljohani, S.L.; Alenazi, M.J. Mpredisn: Multipath resilient routing scheme for sdn-enabled smart cities networks. Appl. Sci. 2021, 11, 1900. [CrossRef]

22. Cascone, C.; Sanvito, D.; Pollini, L.; Capone, A.; Sanso, B. Fast failure detection and recovery in SDN with stateful data plane. Int. J. Netw. Manag. 2017, 27, e1957. [CrossRef]

23. Stephens, B.; Cox, A.L.; Rixner, S. Scalable multi-failure fast failover via forwarding table compression. In Proceedings of the Symposium on SDN Research, Santa Clara, CA, USA, 14–15 March 2016.

24. Lin, Y.D.; Teng, H.Y.; Hsu, C.R.; Liao, C.C.; Lai, Y.C. Fast failover and switchover for link failures and congestion in software defined networks. In Proceedings of the IEEE International Conference on Communications (ICC), Kuala Lumpur, Malaysia, 23–27 May 2016; IEEE: Piscataway, NJ, USA, 2016.

25. Kim, H.; Schlansker, M.; Santos, J.R.; Tourrilhes, J.; Turner, Y.; Feamster, N. Coronet: Fault tolerance for software defined networks. In Proceedings of the 2017 IEEE Conference on Network Softwarization (NetSoft), Bologna, Italy, 3–7 July 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–5.

26. Ochoa-Aday, L.; Cervelló-Pastor, C.; Fernández-Fernández, A. Self-healing and SDN: bridging the gap. Digit. Commun. Netw. 2020, 6, 354–368. [CrossRef]

27. Panev, S.; Latkoski, P. SDN-based failure detection and recovery mechanism for 5G core networks. Trans. Emerg. Telecommun. Technol. 2020, 31, e3721. [CrossRef]

28. Said, S.B.H.; Cousin, B.; Lahoud, S. Software Defined Networking (SDN) for reliable user connectivity in 5G Networks. In Proceedings of the 2017 IEEE Conference on Network Softwarization (NetSoft), Bologna, Italy, 3–7 July 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–5.

29. Lopez-Pajares, D.; Alvarez-Horcajo, J.; Rojas, E.; Asadujjaman, A.S.M.; Martinez-Yelmo, I. Amaru: Plug and play resilient in-band control for SDN. IEEE Access 2019, 7, 123202–123218. [CrossRef]
30. Asadujjaman, A.S.M.; Rojas, E.; Alam, M.S.; Majumdar, S. Fast control channel recovery for resilient in-band OpenFlow networks. In Proceedings of the 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft), Montreal, QC, Canada, 25–29 June 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 19–27.
31. Sharma, S.; Colle, D.; Pickavet, M. Enabling Fast Failure Recovery in OpenFlow networks using RouteFlow. In Proceedings of the 2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), Orlando, FL, USA, 13-15 July 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 1–6.
32. Tomovic, S.; Radusinovic, I. A new traffic engineering approach for QoS provisioning and failure recovery in SDN-based ISP networks. In Proceedings of the 23rd International Scientific-Professional Conference on Information Technology (IT), Žabljak, Montenegro, 19–24 February 2018; IEEE: Piscataway, NJ, USA, 2018.
33. Adami, D.; Giordano, S.; Pagano, M.; Santinelli, N. Class-based traffic recovery with load balancing in software-defined networks. In Proceedings of the IEEE Globecom Workshops (GC Wkshps), Austin, TX, USA, 8–12 December 2014; IEEE: Piscataway, NJ, USA, 2014.
34. Rehman, A.U.; Aguiar, R.L.; Barraca, J.P. Fault-tolerance in the scope of software-defined networking (sdn). IEEE Access 2019, 7, 124474–124490. [CrossRef]
35. Tajiki, M.M.; Shojafar, M.; Akbari, B.; Salsano, S.; Conti, M.; Singhal, M. Joint failure recovery, fault prevention, and energy-efficient resource management for real-time SFC in fog-supported SDN. Comput. Netw. 2019, 162, 106850. [CrossRef]
36. Muthumanikandan, V.; Valliyammati, C. Link failure recovery using shortest path fast rerouting technique in SDN. Wirel. Pers. Commun. 2017, 97, 2475–2495. [CrossRef]
37. Ghanaami, A.; Shao, C. Efficient fast recovery mechanism in software-defined networks: multipath routing approach. In Proceedings of the 11th International Conference for Internet Technology and Secured Transactions (ICITST), Barcelona, Spain, 5–7 December 2016; IEEE: Piscataway, NJ, USA, 2016.
38. Ali, J.; Lee, G.M.; Roh, B.H.; Ryu, D.K.; Park, G. Software-Defined Networking Approaches for Link Failure Recovery: A Survey. Sustainability 2020, 12, 4255. [CrossRef]
39. Tomassilli, A.; Di Lena, G.; Giroire, F.; Tahiri, I.; Saucez, D.; Pérennes, S.; Turlletti, T.; Sadykov, R.; Vanderbeck, F.; Lac, C. Design of robust programmable networks with bandwidth-optimal failure recovery scheme. Comput. Netw. 2021, 192, 108043. [CrossRef]
40. Kiadehi, K.B.; Rahmani, A.M.; Molahosseini, A.S. A fault-tolerant architecture for internet-of-things based on software-defined networks. Telecommun. Syst. 2021, 77, 155–169. [CrossRef]
41. Mohammadi, R.; Javidan, R. EFSUTE: A novel efficient and survivable traffic engineering for software defined networks. J. Reliab. Environ. 2021, 1–14. [CrossRef]
42. Lee, S.S.; Li, K.-Y.; Chan, K.Y.; Lai, G.-H.; Chung, Y.C. Software-based fast failure recovery for resilient OpenFlow networks. In Proceedings of the 7th International Workshop on Reliable Networks Design and Modeling (RNDM), Munich, Germany, 5–7 October 2015; pp. 194–200.
43. Raesi, B.; Giorgetti, A. Software-based fast failure recovery in load balanced SDN-based datacenter networks. In Proceedings of the 2016 6th International Conference on Information Communication and Management (ICICIM), Hertfordshire, UK, 29–31 October 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 95–99.
44. Kuźniar, M.; Peresini, P.; Vasić, N.; Canini, M.; Kostić, D. Automatic failure recovery for software-defined networks. In Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, Hong Kong, 16 August 2013.
45. Yamansavascilar, B.; Baktir, A.C.; Ozgovde, A.; Ersoy, C. Fault tolerance in SDN data plane considering network and application based metrics. J. Netw. Comput. Appl. 2020, 170, 102780. [CrossRef]
46. Chu, C.Y.; Xi, K.; Luo, M.; Chao, H.J. Congestion-aware single link failure recovery in hybrid SDN networks. In Proceedings of the 2015 IEEE Conference on Computer Communications (INFOCOM), Hong Kong, 26 April–1 May 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 1086–1094.
47. Sharma, S.; Staessens, D.; Colle, D.; Pickavet, M.; Demeester, P. In-band control, queuing, and failure recovery functionalities for OpenFlow. IEEE Netw. 2016, 30, 106–112. [CrossRef]
48. Bianchi, G.; Bonola, M.; Capone, A.; Cascone, C. OpenState: Programming platform-independent stateful openflow applications inside the switch. ACM SIGCOMM Comput. Commun. Rev. 2014, 44, 44–51. [CrossRef]
49. Hussain, M.; Shah, N.; Tahir, A. Graph-based policy change detection and implementation in SDN. Electronics 2019, 8, 1136. [CrossRef]
50. De Oliveira, R.L.S.; Schweitzer, C.M.; Shinoda, A.A.; Prete, L.R. Using mininet for emulation and prototyping software-defined networks. In Proceedings of the 2014 IEEE Colombian Conference on Communications and Computing (COLCOM), Bogota, Colombia, 4–6 June 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 1–6.
51. Kaur, S.; Singh, J.; Ghumman, N.S. Network programmability using POX controller. Proc. Int. Conf. Commun. Comput. Syst. (iCCCS) 2014, 138, 134–138.
52. McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Turner, J. OpenFlow: Enabling innovation in campus networks. ACM SIGCOMM Comput. Commun. Rev. 2008, 38, 69–74. [CrossRef]
53. Komajwar, S.; Korkmaz, T. SPRM: Source Path Routing Model and Link Failure Handling in Software Defined Networks. IEEE Trans. Netw. Serv. Manag. 2021, 18, 2873–2887.