Computer Systems
Science and Engineering

Tech Science Press

DOI: 10.32604/csse.2023.034509
Article

Reliable Failure Restoration with Bayesian Congestion Aware for Software Defined Networks

Babangida Isyaku[1,2,*], Kamalrulnizam Bin Abu Bakar[1], Wamda Nagmeldin[3], Abdelzahir Abdelmaboud[4], Faisal Saeed[5,6] and Fuad A. Ghaleb[1]

[1]Faculty of Computing, Universiti Teknologi Malaysia, Johor Bahru, 81310, Malaysia
[2]Faculty of Computing and Information Technology, Sule Lamido University, Kafin Hausa, P.M.B.048, Nigeria
[3]College of Computer Engineering and Sciences, PrinceSattam bin Abdulaziz University, Al-Kharj, 11942, Saudi Arabia
[4]Department of Information Systems, King Khalid University, Muhayel Aseer, 61913, Saudi Arabia
[5]Taibah University, P.O. Box 344, Medina, 41411, Saudi Arabia
[6]School of Computing and Digital Technology, Birmingham City University, Birmingham, B4 7XG, UK
*Corresponding Author: Babangida Isyaku. Email: isyaku@graduate.utm.my
Received: 19 July 2022; Accepted: 21 November 2022

Abstract: Software Defined Networks (SDN) introduced better network management by decoupling control and data plane. However, communication reliability is the desired property in computer networks. The frequency of communication link failure degrades network performance, and service disruptions are likely to occur. Emerging network applications, such as delay-sensitive applications, suffer packet loss with higher Round Trip Time (RTT). Several failure recovery schemes have been proposed to address link failure recovery issues in SDN. However, these schemes have various weaknesses, which may not always guarantee service availability. Communication paths differ in their roles; some paths are critical because of the higher frequency usage. Other paths frequently share links between primary and backup. Rerouting the affected flows after failure occurrences without investigating the path roles can lead to post-recovery congestion with packet loss and system throughput. Therefore, there is a lack of studies to incorporate path criticality and residual path capacity to reroute the affected flows in case of link failure. This paper proposed Reliable Failure Restoration with Congestion Aware for SDN to select the reliable backup path that decreases packet loss and RTT, increasing network throughput while minimizing post-recovery congestion. The affected flows are redirected through a path with minimal risk of failure, while Bayesian probability is used to predict post-recovery congestion. Both the former and latter path with a minimal score is chosen. The simulation results improved throughput by (45%), reduced packet losses (87%), and lowered RTT (89%) compared to benchmarking works.

Keywords: SDN; OpenFlow; failure restoration; critical path; Bayesian probability

1 Introduction

Software Defined Networks (SDN) is an emerging network paradigm that fosters network innovation by separating the controller from the data plane using the OpenFlow standard [1]. This separation improved the network management, increased the chance of deploying innovations, and made the network more programmable [2,3]. However, networking equipment is failure-prone; therefore, some aspects such as reliability and fault management are necessary properties in the computer network. The frequency of communication link failure on the data plane has been one of the challenging issues. On average, every 30 min, some links are likely to fail [4]. Another study reported the probability of having one link failure within a 5-min cycle by more than 20%, which is not negligible [5]. Therefore, upon occurrences of failure, the location of the failure is detected, and the failure event is communicated to the SDN controller. In return, it computes available alternative paths with a set of forwarding rules and instructs the affected switches to install the rules on their Flowtable to enable the forwarding of affected traffic flows. Facilitating reliable failure recovery depends on how fast the network is reconfigured through a path with minimal chances of future occurrences. Failure recovery can be achieved using a restoration or protection approach. In the former, the backup path is computed after failure detection. Unfortunately, frequent flow arrival can easily affect the performance of the controller. There can be up to 200,000 flows arrival/sec for a data centre with a 4 k server [6]. Calculating the end-end new backup path for this dense number of flows can easily overwhelm the SDN controller. In addition, when the number of flows exceeds the residual capacity of the transmission links, link quality will be affected by a lower transmission rate. The latter provisioned backup paths forwarding rules in advance in either of the three methods to overcome these challenges. (i) one backup path can be deployed to protect exactly one primary path. The work in [7] is an example of such a scheme. (ii) Another possibility is to configure one backup path to protect N number of primary paths [8]. (iii) It is also possible for specified Primary (P) paths to be protected by N paths, such that $N \leq P$, like the works in [9,10]. In either case, many forwarding rules are deployed in the switch Flowtable. It is a logical data structure in SDN switches that guide how flows are handled. Switch Flowtable is implemented using high-speed memory, i.e., Ternary Content Addressable Memory (TCAM). However, the speed is at the cost of limited space [11]. Although, the presence of backup path forwarding rules in advance can help to locally reroute the affected flows without consulting the controller. Unfortunately, deploying many forwarding rules to protect these dense number of flows can easily overflow the precious memory resource. Conversely, removing the affected flow rules in the set of switches and updating the new routing rules for the path may considerably lead to significant packet delay and losses. In line with the Carrier Grade Networks (CGN), the recovery process must be completed within 50 ms to comply with the Service Level Agreement (SLA) of Internet Service Providers (ISPs). Similarly, average packet loss should be within 0.3% with 99% port service availability [12]. Therefore, time and memory space are the major challenge for both restoration and protection failure recovery approaches. Although, the protection approach can quickly enable alternative paths. However, it may be more effective offline where the network setting is fixed. Unfortunately, traffic flows change more frequently, especially with the emergence of Internet of Things (IoT) applications. As such, it may not cope with real-time applications. This way, the restoration approach received attention lately. Backup path forwarding rules allocation is considered a challenging task using restoration [13]. Inappropriate backup path selection may fail earlier than the primary path. Moreover, the density of network topology differs and when the backup path fails, it may affect multiple paths which this paper referred to as path criticality, and consequently decrease service availability. Similarly, the links along the backup path may get overloaded leading to

congestion and eventually packet loss with higher recovery time. Therefore, various recovery schemes have been proposed to reduce the recovery time [4,14,15].

Other solutions focus on selecting backup paths while optimizing the number of flow rule operations [14,16,17]. Another solution established an alternate route to handle high-priority packets with minimal delay [18]. Path partition was among the recent solutions to reduce end-end path computation [13]. However, a prior study [19] reported that 30% of link failures occur due to multiple shared links referred to as critical links on the path. Consequently, decrease in service availability. Based on the above discussions, there is a lack of study to properly incorporate links path criticality and post-recovery congestion to improve the service availability and decrease the number of packet losses, thereby reducing packet loss, RTT, and increasing throughput. This paper proposed Reliable Failure Restoration with Congestion (RFRC) to select the reliable backup path that decreases service disruption, increasing network service availability while minimizing post recovery congestion. This is achieved by rerouting the affected flows to the backup path that is least likely to share the link between primary and backup path. This choice has minimized the probability of failure between more than one paths. The contributions of this paper can be summarized as follows.

- This paper discussed the weakness of the existing failure recovery approaches in SDN
- The paper devised a model to estimate path criticality model based on the frequency usage of the set of links on the path. The failure restoration approach is embedded with this model to redirect the affected flows after the occurrence of link failure.
- The study introduced the Bayesian probability post-recovery congestion model to examine links state. The model predicts a path with minimal post-recovery congestion if there is a share link between the primary and the selected backup path.
- An experiment was conducted to evaluate the performance gain of the proposed scheme compared to the existing methods, which shows the proposed RFRC has superior performance.

The rest of the paper is organized as follows: Section 2 presents the various failure recovery-related works. The design of the proposed solution is described in Section 3, and experimental setup and performance evaluation are presented in Section 4. Finally, Section 5 concludes the work and suggests future work.

2 Related Works

This section discusses and investigates the existing failure recovery schemes for SDN. Presently several recovery schemes have been proposed from the past literature. These works are categorized into restoration and protection to detour the affected flows upon link failure. The following subsections details each scheme.

2.1 Restoration Approach

Some works rely on restoration approaches to detect a link failure and establish an alternative path. References in [20–22] are an example of such an approach. They calculate the optimal backup path after failure detection and reroute the affected flows using the restoration approach. However, the recovery time is approximately 100 ms, which is far from the CGN requirement. The works in [23,24] argued that periodic link monitoring to detect the failure before establishing a backup path might considerably introduce controller overhead, leading to higher recovery time. To overcome these challenges, [24] offload the link monitoring capability of Operation Administration and Maintenance (OAM) from the controller to the OpenFlow switch. OAM leverages on general message generator and processing function in the switches, and extension in OpenFlow 1.1 protocol to support the

monitoring function. However, delegating some of the control functions to switches violates SDN's promise [3]. Alternatively, the work in [25] presents a fault management scheme without modifying the SDN architecture. In this regard, a topology discovery module was devised to collect the link-state event periodically. Afterwards, the route planning module used the gathered information to calculate multiple route paths based on the topology information. Upon failure, the A Virtual Local Area Network (VLAN) switch configuration module configures multiple switch ports with relevant VLAN Identifier (IDs) to enforce each routing path. However, forwarding rules update operations were overlooked. Consequently, it affects the recovery time. Multi-Failure Restoration with Minimal Flow Operations has been proposed in [17,26]. The authors devised an optimization scheme to find a restoration path with a Dijkstra-like path cost and minimum operation cost. The challenge is to find a reroute path achieving a trade-off between operation cost and end-end path cost [16]. An end-to-end fast link failure recovery approach based on the shortest was introduced in [18]. The scheme categorized packets, and high-priority packets are redirected to an alternative path upon failure occurrences. However, the solution is tested in a small network setting, which may not be feasible in a large-scale network, because the algorithm's complexity augments as the network's size increases [13].

Similarly, the work in [27] devised a scheme for the SDN controller application to classify flows according to their protocols. This way, the scheme obtains the Flow Table utilization ratio and computes the backup path with sufficient bandwidth. Upon the occurrence of failure, the scheme examines the residual capacity of switch resources and computes alternative paths accordingly. The work in [28] proposed a mechanism to avoid frequent contacting the controller and take local corrective measures. In this regard, two methods were devised to store bypass paths on all pairs of nodes and others on some selected nodes. Therefore, when a failure occurs, a switch can act locally. However, installing two sets of flow rules bypass in the switch Flowtable beside the primary path rules will lead to load in balance and caused congestions.

Other solutions attempt to reduce the end-to-end computation and Flowtable time. For example, the works in [29,30] introduced the principle of a community detection scheme. If a failure occurs, the affected community is detected, and the backup path is established within the affected community without tampering with packet forwarding in other communities. This way, recovery is faster, thereby improving the network fault tolerance capability. However, removing the old flow entries in the affected path and re-installing the new entries for the alternative can be costly, especially when the path length is long. Toward this goal, their follow-up works [7,13,30] further reduced the update operation mainly cost as a result of old flow entries. When failure occurs, the scheme only searches the new path from the point of failure down to the destination switch and removes the old flow entries of the affected switches only; the remaining flow entries on the path are preserved. This way, the scheme has significantly reduced the update operation and end-to-end computation time. However, the scheme neither guarantees the shortest path nor considers congestion.

In contrast, an effective method to minimize the total switching time of all paths in the network was presented in [31,32]. The authors considered path route selection based on path switching latency in heterogeneous networks, where switches had different specifications. This way, several paths are explored, and the path with the set of switches having the shortest processing time is considered. However, the solution may not give optimal performance in a large-scale network.

2.2 Protection Approaches

In the protection mechanism, rules must be preinstalled in advance for path and local recovery approaches. Due to the proliferation of network flows per second, the number of rules in the switch

flow table may quickly escalate, leading to large switch memory TCAM consumption. To minimize the TCAM consumption, the references [4,33] proposed a set of algorithms: Forward Local Re-routing (FLR) and Backward Local Rerouting (BLR) to compute backup paths for a primary route for faster failure recovery. Local re-routing of the failed traffic from the point of failure enables speedy recovery. FLR and BLR backup paths improved sharing of forwarding rules at the switches thereby choosing backup path with least number of additional switches. However, the solution neither considers post-recovery congestion nor the potential future failure of the selected link. The works [34–36] leverage VLAN tagging to present a new protection method to aggregate the disrupted flows. This way, several flow rules are reduced, thereby improving the recovery time of carrier-grade recovery requirements. However, the aggregation technique may reduce flow visibility which in turn may be challenging for the controller to maintain the global view of the network. Their fellow up work implements two algorithms; Local Immediate (LIm) recovery strategy, in which the controller will utilize the fast failover to locally switch to an alternative path using VLAN tagging feature to tag the arriving packets with the outgoing link ID. While Immediate Controller Dependent (ICoD) recovery required controller intervention to establish alternative path. This way, recovery time may be faster because of the fast failover, which allowed a quick and local reaction to failures without the need to resort on central controller.

Various attempts were made to achieve faster recovery using fast failover to avoid controller involvement in detouring the affected flow. The reference in [37] proposed Fast Failover (FF) link failure with a congestion-aware mechanism. FF is preconfigured with multiple paths to redirect the disrupted flows to a failure state. Based on the FF configuration, the controller periodically monitors the status of the switch port to perceive the failure on time. The protection scheme resulted in an average recovery time of around 40 ms. However, constant controller monitoring can introduce extra processing load on the controller. A scalable multi-failure FF was presented by [38]. Their work dynamically compressed the alternate path's flow entries of the incoming flows with the existing flow entries on the backup path. In this way, the total number of rules would be significantly reduced. However, such a dynamic procedure may lead to an extra processing load on the controller to configure the primary and backup path for every new arrival flow.

Moreover, the number of backup path rules augments as the flow arrival increases [39]. An efficient fault-tolerant memory management aware approach was proposed [34]. The scheme computes path protection per link instead of rules per flow by configuring VLAN tagging for each link failure. This way, A VLAN tagging is provided for each link identification while defining backup path rules. Therefore, reducing the number of forwarding rules would be proportional to the network setting. However, in a large-scale network, the overhead would be non-trivial [15]. Another solution [40] considered the switch Flowtable storage constraints to devise a recovery scheme. A Fault-Tolerant Forwarding Table Design (FFTD) was introduced to group the flows using group entries and aggregates the flows using a tagging mechanism for rapid recovery from the dual failures. This way, FFTD satisfies the GCN's 50 ms recovery requirement, reducing the backup path flow storage requirement. However, neither [34] nor [40] considered post-recovery congestion after the localized recovery. A shared ring was proposed to reduce the consumption of a backup resource [9]. The authors devised a ring-based single failure recovery approach to reduce the number of entries.

A ring circle in the network topology is selected to act as a share backup path, based on the all-backup path to improve the Flowtable utilization. Although the approach could improve the recovery time and backup resource consumption, network post-recovery congestion may occur. Efficient congestion and memory-aware failure recovery (SafeGuad) were presented [5]. SafeGuard iterates through a backup path of the impacted flows to ensure that the rerouting switches have enough

space to accommodate the backup path rules. Residual link capacity is checked to avoid post-recovery congestion. This way impacted flows are deployed efficiently. However, Safeguard may be expensive because it requires two paths to be installed for each flow, which could overwhelm critical network resources such as switch TCAM [13].

Based on the previous discussion, different studies have been proposed to address the communication link failure on the SDN data plane. Some techniques manage flow rules to improve failure recovery, while others deal with restoration latency issues. Protection may have better performance at the cost of significant memory space. In addition, it may not always cope with the real-time nature of traffic variabilities. Restoration can cope with real-time traffic flows arrival but at the cost of controller overhead. This study focused on the restoration approach. Some works investigate the restoration approach in a small network setting. However, in large-scale networks, path importance differs; some paths are critical because of the number of shortest paths that pass through them. Other links on paths are frequently shared between primary and backup paths. Failures on either path may lead to failure on multiple paths. Based on the above discussion, there is a lack of studies to incorporate path criticality and post-recovery congestion to reduce packet losses and improve throughput. While research is better with time, SDN fault management still needs more research and investigation.

3 Design of the RFRC Scheme

The working procedure of the Reliable Failure Restoration with Congestion (RFRC) scheme is illustrated in Fig. 1. The topology manager triggers a periodic monitoring mechanism to discover the network entities at time t. This includes the hosts, switches, and the corresponding links between the switches. Afterwards, the network is built, and K paths are computed based on dynamic link-state information. RFRC periodically check the network condition; flows are transmitted through the primary path under normal network condition. However, when a link failure occurs, the failure detection manager detects the location and shares the failure event with critical and congestion detection modules to examine the available path from the failure location. As explained in Section 3.3, the critical path detection phase adds critical value (P_{c_value}) and appends the value to each path in K. Similarly, the congestion detection phase examines the set of paths as explained in Section 3.5. This information is forwarded to the reliable backup path routing at the failure restoration engine to choose the right backup path when a failure occurs. This way, the backup path selection manager rerouted affected flows based on the real-time network situation. The following section explains each phase in detail.

3.1 Network Model

Let graph G (N, L) represent the network topology, where N denotes a set of switches and $L_{i,j}$ is a set of bidirectional links. Let $x = |N|$ be the number of switches and $y = |L|$ be the number of links in the network. Each link $(i, j) \in L$ is associated with a weight ($w_{i,j}$,) based on the Dynamic link-State Information (DLI). A failure on link $L_{i,j}$ also affects $L_{j,i}$. A path is a sequence of switches with members of L. Since traditionally paths are computed with static Link State Information (LSI) (i.e. shortest P), the paper begins with a set of possible paths from *src* to *dst* $P = \{P_{s,t,}\}$ using shortest P $= min \sum_{s}^{t} w_{i,j}$. However, computing P based on LSI may not yield optimal performance, an additional constraint is necessary to meet the demand of real-time applications. As such, RFRC incorporates DLI in the computing set of P. This way, the set of possible P are computed with different link state value. Unfortunately, computing all possible P may grow quickly with an increase in network size. Which in turn may add more computational load on the controller. This issue is addressed by restricting the

choice of P to k shortest path candidates, where $k = 5$. Afterwards, the criticality value of each $p_i \in P$ is estimated and sorted by their criticality values. During normal network operation, flows are routed through $P_{s,t} = \{^{R}{}_{i,j}\}$ as shown in Eq. (1).

$$R_{i,j} = \begin{cases} 1 \ P \ traverse \ through \ l_{i,j} \\ 0 \ otherwise \end{cases} \tag{1}$$

It is assumed $\{^{R}{}_{i,j}\}$ is a path with no cycle, and the link $\{l_{i,j}\}$ operational state over the route is defined by $^{OP}{}_{i,j}$ as presented in Eq. (2). The link operational state is derived from Eq. (2) which expresses that for a link to declare un operational, it must be part of the selected path to route flow from source to destination. Afterwards, it must not be idle as presented in Eq. (3).

$$OP_{i,j} = \begin{cases} 1 \ operational \\ 0 \ otherwise \end{cases} \tag{2}$$

$$F_{i,j} = \left\{ l_{i,j} | l_{i,j} \in L \wedge \left(R_{i,j}(P) = 1 \right) \wedge \left(OP_{i,j}\left(l_{i,j} \right) = 0 \right) \right\} \tag{3}$$

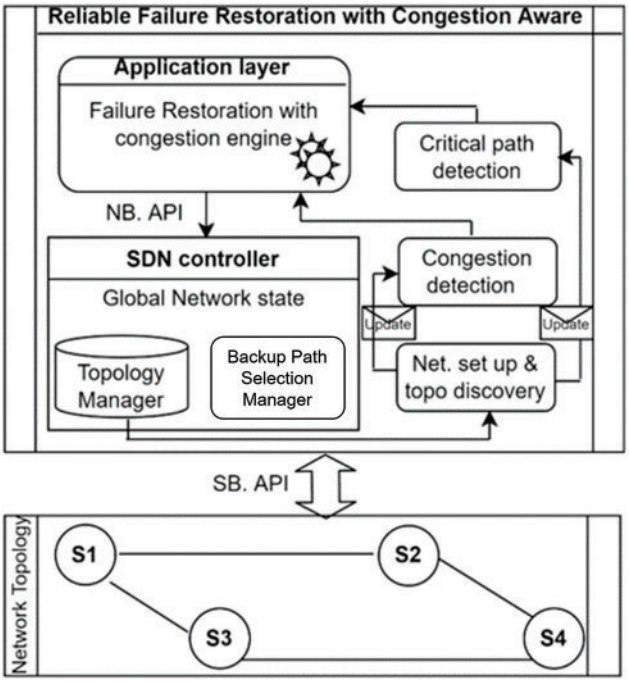

Figure 1: Architectural design of RFRC

3.2 Link Failure Operation

The RFRC is initiated with the route discovery phase, where the SDN controller discovers the network forwarding elements and builds the network, as explained in the previous section (A). Afterwards, during normal network operation, traffic flows are transmitted through an optimal primary path. This way, RFRC periodically examines the network link state, once failure occurs, the location is detected, and path examination begins. The proposed solution considered two link operational events to ensure reliable backup path selection with less congestion.

Referring to the first event in Fig. 2, source nodes S1 and S2 are sending packets to destination node D. It can be observed there are different paths, P1 and P2 have slightly different numbers of hops while P3 have minimum hops, and there is a share link among the three paths indicated as a solid line. Since the primary path usually has the minimum number of hops, it is assumed P3 served as the primary path. Therefore, failure on the share link (S2–S4) will significantly affect the three paths, consequently, lead congestion and sometimes even service disruption. In this regard, an effective failure recovery design shall take care of such a situation. Assuming the dotted line in Fig. 4 indicates the primary (S2–S4–D). If a failure occurs on link LS4–D, there would be at least three alternative paths from the detect node (S4), namely P4: (S4–S7–S10–D), P5: (S4–S5–S9–S10–D), and P6: (S4–S8–D). Also assumed, three flows are generated with different sizes (number of packets). These flows are affected and require rerouting through a backup path. All the affected flows would compete for the shortest path (P6) regardless of size. In this case, P6 is referred to as a critical path. Since backup paths may fail earlier than primary, failure on the critical path may lead to service disruptions. Moreover, it is crucial to understand each failure scenario might have different consequences and the severity of adverse effects of each failure scenario varies according to the flow types. For example, if F1 passing through P1 contained delay-sensitive applications such as VoIP and F2 passing through P2 containing best-effort flows. The consequences of F1 might be different from F2 during the failure state. Therefore, both share link and critical path issues are addressed in this research. Firstly, a set of available paths from the detect node are examined, if there exists a shared link between the primary and backup path, Bayesian probability predicts the residual path capacity and chooses the path with enough capacity to reroute the affected flows. Otherwise, critical paths are detected before rerouting the affected flows. For example, P4 and P5 are examined based on topological knowledge using edge betweenness after failure detection. Fig. 3 illustrates the design, while the following sub-section explains the procedures.

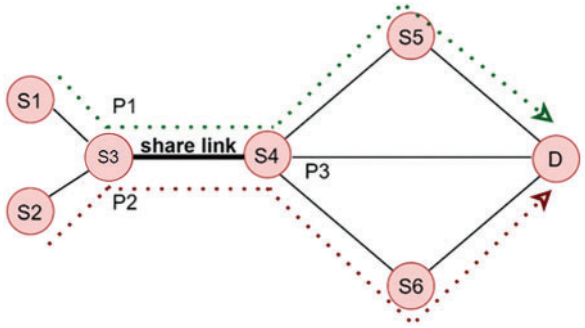

Figure 2: Share link scenario

3.3 Path Criticality

Primary and backup paths required a reliable path with the least multiple paths correlation, in other words, the least correlated potential future failure $^{p}{}_{fo}$. To this end, the double links failure at a slight time is derived according to the approximation of the multiple link failures probabilities Eq. (4).

$$\Pr\left(P_m^{st}, P_n^{st}\right) = \sum_{(i,j)\in P_m^{st}} \sum_{(x,y)\in P_n^{st}} \Pr\left(l_{i,j}, l_{x,y}\right) \tag{4}$$

The approximation in Eq. (4) serves as a guide to calculating the multiple path failure probabilities with insufficient information. Conversely, path criticality estimates multiple potential links failures probabilities based on the knowledge of the global network topology using Edge Betweenness Centrality (CB). $CB\left(l_{i,j}\right)$ is used to reflect the importance of a link with others in a network's topology

graph since switches forward traffic along the shortest cost paths. Therefore, it measures the number of potential correlated failures on a particular link. This way, the number of paths between *s*-*t* that pass γ (*s, t*) and the number of shortest paths between *source* to *destination* passing through link $L_{i,j}$ γ (*s, t*/$l_{i,j}$). The shortest path edge betweenness centrality $CB(l_{i,j})$ of an edge *e* is derived from Eq. (5).

$$CB_{(l_{i,j})} = \sum_{(s,t)\in N} \frac{\gamma\,(s,t|\,l_{i,j})}{\gamma\,(s,t)} \tag{5}$$

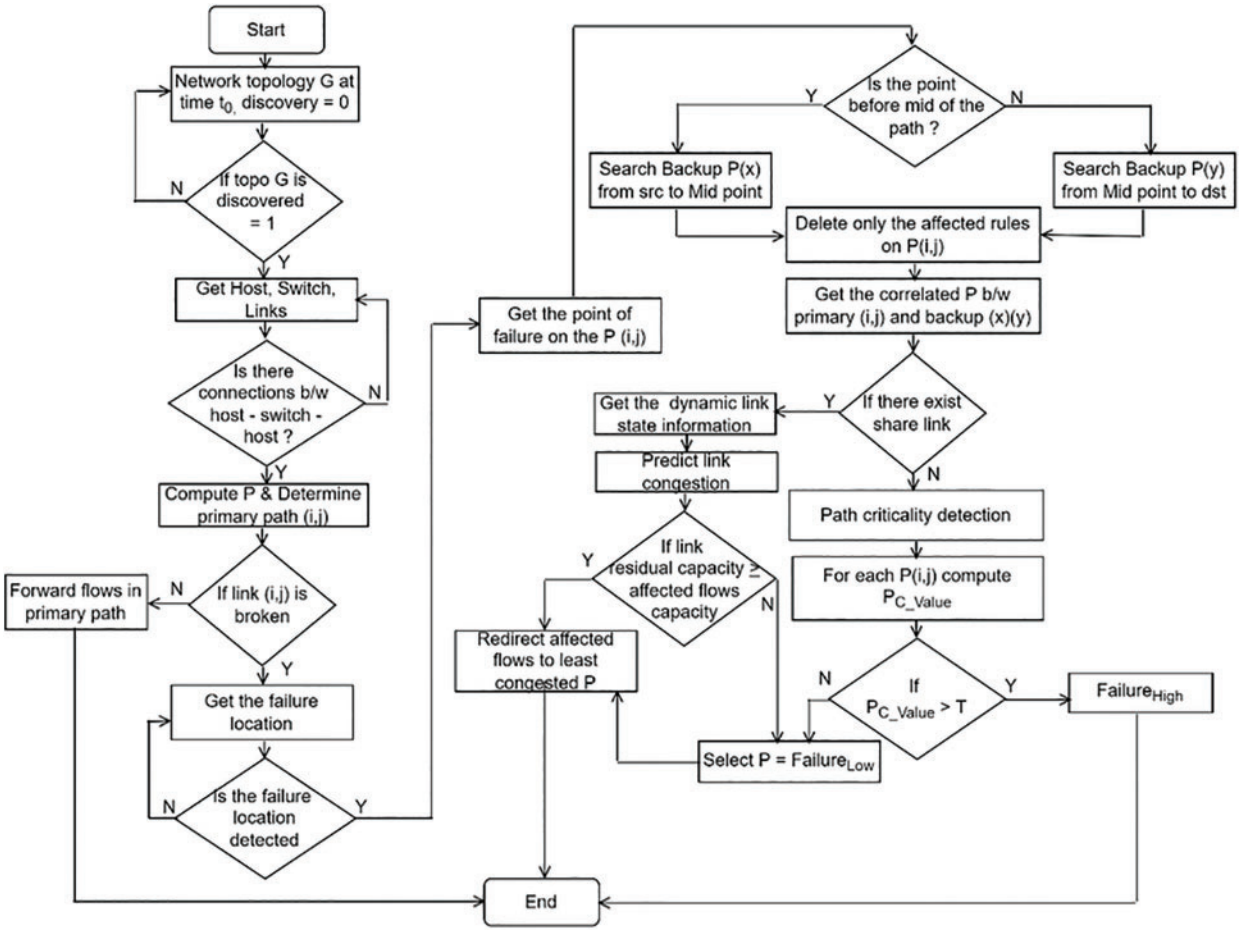

Figure 3: Failure restoration procedure

Hence, at time t_1 after the discovery process, RFRC iterates through all the network links to obtain the $CB(l_{i,j})$ for each link on the path, as presented in Eq. (6).

$$L_d\,\{(i,j)\,,(x,y)\in L,(i,j)\neq(x,y)\} = CB(l_{i,j}) \tag{6}$$

This is done by the Path Computation Element (PCE), which computes the *k*-shortest paths (*k*-SP) between $N_{(S)}$–$N_{(T)}$. Thereafter, the RFRC iterates through all $l_{i,j}$ on $p_i \in P$ to obtain the summation of the path criticality value, which in turn is used in evaluating the criticality of each p_i refer to Eq. (7).

$$P_{c\_value} = \sum_{(i,j)\in L} (\forall i,j|L_{d1} + L_{d2}\ldots n.n \leq len(p)) \tag{7}$$

where *i, j* represents the link on the candidate path, and L_d indicates the criticality value for that link until the last link on the path as *n*, for every *n* should be less than equal to the total path length. This way, the module will return the summation of the associated critical value (Pc_value) to P. It is possible for more than one P with the same P_{c_value}. In this regard, the RFRC will select the path with minimum distance. It is expected that when P has P_{c_value} < Threshold value, in such case, correlated failure is least likely to happen, given by Eq. (8). The reliability of such a path will be increased, and the performance of data transmission will not be reduced too much when the current path is broken. Because of the least number of path correlations which translates to the least number of potential failures pfo, this is the particular interest of RFRC.

$$\Pr = \left(l_{i,j} | l_{i,j(CB)} < T \wedge x, y\,(CB) = T\right) if\ (i,j) \neq (x,y) \tag{8}$$

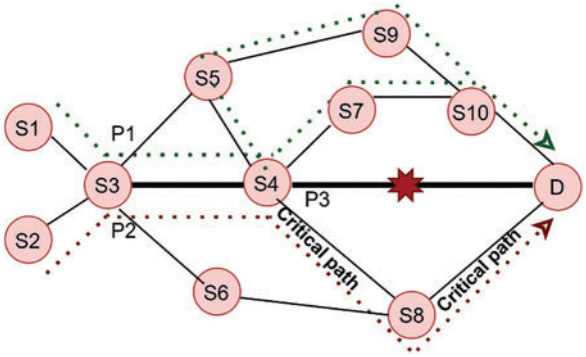

Figure 4: Critical path scenario

3.4 Removal of Affected Flows

When the failure event occurs $\{^{F}{}_{i,j}\}$ the upstream node detects the failure, tags the packet and sends operational link status $(^{OP}{}_{i,j}\ (l_{i,j}) = 0)$ to the controller. In return, action is required to restore the link operational state to $(^{OP}{}_{i,j}\ (l_{i,j}) = 1)$. The controller flushes out the old flow rules on the affected primary path presented $^{x}{}_{fr}$ Eq. (9). However, deleting all the flow forwarding rules in the affected routes can significantly result in higher update operations which contradicts the aim of this study. Assuming the path length is long, many rules will be removed in this situation, and new forwarding rules will have to be installed.

$$Flush\_x_{fr} = \left(R_{i,j}\,(P) = 1\right) \wedge \left(OP_{i,j}\left(l_{i,j}\right) = 0\right) \tag{9}$$

In contrast, the RFRC gets the affected switches, identifies the port number associated with the affected link $\{l_{i,j}\}$ and removes that particular forwarding rule. In this way, two forwarding rules are removed, derived in Eq. (10). The procedure is demonstrated in algorithm 1 line (1–12).

$$R_{old\_remove} = \left\{d_{pid} \in N \wedge p_{i,j} | p_{i,j} \in port_{no} \wedge F_{i,j}\right\} \tag{10}$$

Thereafter, Link Layer Discovery Packets ($LLDP_{pkt}$) are broadcasted in the network topology to update the updated topology information. Once the topology information is obtained, the route for the final selected backup $\{^{y}{}_{br}\}$ will be computed considering path criticality while chosen path with the slightest chance of post-recovery congestion.

Algorithm 1: Removing old, affected rules

```
    Input: G (N, L)
1   If F_{i,j} = (OP_{i,j} (l_{i,j})= 0) Do
2     If {s_i, s_j}∈ F_{i,j} ∧{s_i, s_j}∈ mid(P)
3       If s_{i,j} port state == 1
4         F_List = {s_i, s_j}
5         // check which location do affected s_{i,j} lies on
6         If F_List [0] ∧F_List [1] in c_i: c_i⊆ G
7           Get the affected (c_i)
8           // clean up the route x_fr in s_{i,j} with target port_no
9           Get s_{i,j} and port_no in c_i: i,j ∈ N
10          For (s_{i,j}) in F_List
11            Rebuild G.
12            R_{old,remove} old rules s_{i,j}
13          end
14        end
15      end
16    end
17  end
```

3.5 Bayesian Post-Recovery Congestion Avoidance

Blindly redirecting traffic flows on the failed link to a new reroute path would cause potential congestion in the post-recovery network. Therefore, it is crucial to avoid post-recovery congestion to improve overall clique performance. Since the importance of the paths is not the same, other paths may always carry a high amount of traffic flow because it is being frequently used; such paths tend to have a high load all the time. This can be interpreted as some sets of links in the path are likely to experience congestion. The least loaded routing in this scenario refers to selecting p_i with a minimal sum of links load, which may likely not be congested during post-recovery. In SDN, the controller periodically pulls the statistical information about the total bytes passed through each switch link port and stores this information to calculate links load. This way, RFRC leverage such info to measure the link load on each path and thereafter predict the probability of a congested path. In this regard, Baye's decision rule will effectively test this scenario. Knowing the likelihood and the prior is crucial to arrive at the determined probability class. To this end, the probability of path congestion is proportional to the sum of $l_{i,j}$ load $l_{i,j} \in \mathrm{P_i}$, $\mathrm{Pl_{i,j}}(\delta)$. Thus, the prior probability is derived from the average link load event with the total capacity of the link on the path which is derived from the total cost of each path $p_i \in$ P defined by Eq. (11). In this case, an event is an attempt to redirect the affected flows $\{^{x}_{fr}\}$ to a path, where the residual capacity of set of links on the path may or may not have the sufficient capacity to accommodate the number of $^{x}_{fr}$. The likelihood of the high load on a set of links $(l_{i,j}) \in \mathrm{p_i}$, given the information about the probability of congestion, δ, is defined as Eq. (12).

$$xfc_i = \left(\sum\nolimits_{(i,j)\in L}^{N} (w_{i,j}) + yvc_i = \sum\nolimits_{(i,j)\in L}^{N} \frac{l_{i,j}}{l_{(i,j),c}}\right) \tag{11}$$

$$\mathrm{P_r} = \mathrm{Pl}_{i,j}(x = congestion|\delta) = xfc_i \tag{12}$$

where xfc_i defined the total predetermine cost for each link derived from the link fixed weight $w_{i,j}$ cost such that $l_{i,j} \in$ pi, and the link load $l_{i,j}$ bps is obtained from the statistical information. The yvc_i represents the variable cost of link $l_{i,j,}$ which is derived from the recent changes in byte count for all flow

entries in a switch (S_i), while $l_{(I,j),c}$ shows the total capacity of the link. Therefore, upon occurrences of failure, if RFRC wants to redirect the affected flows of $\{^{x}_{fr}\}$ through a set of links $l_{i,j}$ on $p_i \in P$, the RFRC can update the uncertainty that the set of links load are congested or otherwise, using Bayes rule, $P_{li,j}(\delta|P_r)$, which give the greater posterior probability of the path congestion occurrences. This way, without knowing the frequency of the congestion initially, by redirecting the affected flows $\{^{x}_{fr}\}$, the posterior probability can be updated about the congestion on the link, given this sequence of observed features using Bayes law, as derived from Eq. (13).

$$P_{I,J}(\delta/\tau) = \frac{P\left(\frac{\tau}{\delta}\right) P(\delta)}{P(\tau)} \tag{13}$$

Algorithm 2: End-to-end backup path selection

Input G (N, L)
1 // compute end-to-end set of $p_i \in$ P with least critical links
2 Set of P=$\{p_1, p_2, \ldots p_n \in P\}$
3 **For** p_i in P
4 Assign each p_i its P_{c_value}
5 **end**
6 //Finding p_i with the least set of critical links
7 $min(p_{s,t}) = 0$
8 **For** p_i in set_P
9 P_criticality = get(Set of P)
10 **If** $min(P_{s,t}) = 0$
11 $min(P_{s,t})$ = P_criticality
12 *Continue*
13 **end**
14 **If** P_criticality < $min(P_{s,t})$
15 $min(P_{s,t})$ = P_criticality
16 $min(P_{s,t}) = p_i$
Return P_i
17 **end**
18 **end**

Therefore, according to Bayes' theorem, the returned outcomes tell the posterior probability of which p_i among the set of $p_i \varepsilon P_{s,t}$ may likely be congested. The p_i with the total sum of residual link capacity greater than equal to Threshold (T) would be the best candidate to reroute the affected flows $\{^{x}_{fr}\}$ toward its destination, where T is usually set as the capacity of the affected flows. This way, the chance of having a high delay or packet loss is reduced, consequently leading to increased throughput.

3.6 Reliable Backup Path Routing

Initially, dynamic traffic matric was generated at different times t defined in Eq. (14). Where T_{flow} represents traffic flows types which include TCP, UDP, and T_{fd} indicating the traffic duration, while T_{fv} corresponds to the traffic volume and T_{ps} signify the traffic volume variation per second from source to destination demand pairs. A set of k shortest paths P = { p_i: $p_i \in K$} was obtained, and the critical P_{value} for each p_i was also estimated through a set of links on p_i. Afterwards, the summation of p_{value} from source s to destination switches $\sum_s^t p_{value}$, Which in turn make up the path p_i critical value as shown in Algorithm 2 lines (1–5). Therefore, efficient routing is triggered based on two situations: During

normal network operation and upon arrival of new flows. Secondly, when a failure event occurs. For the latter, several random failures at different time intervals were generated at various locations on the selected path. Afterwards, the SDN controller received a failure event notification; two operations are carried out in this regard. Firstly, the location of the failure is detected. Afterwards, the scheme removed the affected forwarding rules $\{^{x}_{fr}\}$ for the old flows in the switch Flowtable of the affected switches to clear space for new rules using R_{old}. The procedure is demonstrated in Algorithm 1 line (1–12). Secondly, a backup path will be computed for the affected flows.

In this regard, RFRC examines if there is no share link between BP (x, y) and $^{R}_{i,j}$. Potential correlated failures are checked through link criticality. The path with the least critical links is chosen as demonstrated in Algorithm 2 lines (6–18). However, if shared links exist, RFRC leverages Bayesian probability to predict a less congested path to avoid the occurrences of post-recovery congestion, as indicated in Algorithm 3, lines 12–15. For all the links on the Path, RFRC obtains the link state information to make an informed decision Algorithm 3 lines 16–19. If the residual link capacity is greater than the capacity of the total number of affected flows, that path is chosen to redirect the failed flows, Algorithm 3 lines 20–23. Otherwise, a random path with the least critical links is chosen in Algorithm 3, lines 23–26. Finally, the correspondent forwarding rules of the returned path are installed on the set of switches as described in Algorithm 3, line 23.

$$TM_{st}(t) = \left(\left(T_{flow}\right) + \left(T_{fd}\right) + \left(T_{fy}\right) + \left(T_{ps}\right)\right) \tag{14}$$

Algorithm 3: Reliable backup path routing

```
1   Procedure (G, N, L, K)
2   G represents the graph topology of the network
3   N represents switches/nodes in the network
4   K represent set of paths = {P_i ... P_k}
5   // check the network's operational state
6   If OP_{i,j} ==1 do
7      Forward f to primary P /* Eq. (1) */
8   Else
9      F_{i,j} = (OP_{i,j} (l_{i,j})= 0) do
10     Get the location of the failure
11     Recall algorithm 1 to remove the corresponding rules in Flowtable
12     //examine the path correlation between primary and backup
13     If there exists a share link b/w primary and backup
14        Get DLS information
15        Predict congestion
16        For ∀ link ℓ_{i,j} in P_i: P_i ∈ K
17           P_i {ℓ_{i,j}} = cost of set of link capacity /* Eq. (11) */
18           P_i {ℓ_{i,j}} = congestion probability /* Eq. (12) */
19           P_i {ℓ_{i,j}} = update posterior probability /* Eq. (13) */
20        If P_i residual capacity > the capacity of x_fr
21           R_install P_i
22           Redirect x_fr on p_i ∈ K
23        Else
24           Select random P_i= min {P_{s,t} = P_{c_value}} /* Eq. (7) */
```

(Continued)

Algorithm 3: Continued

```
25              R_install P_i
26          End
27      Else
28          Recall algorithm 2
29          Get P_i
30          Route the affected flows x_fr
31      End
32  End
```

3.7 Computational Analyses

This paper presents RFRC with three modules (critical path detection, post-recovery congestion avoidance, and backup reliable path routing). The computational complexity of RFRC depends on the complexity of the three modules. Critical path: Initially, it obtains the number of switches (S), link numbers (E), and number of flows (M), computes *k* paths among the switches (K), and appends their critical value (P_{value}), and its complexity is O(SEMK).

Post-recovery congestion regularly monitors the network, obtains the link load, and sums it up to get the path capacity (P); the complexity is O(EK). The module subtracts the number of affected flows from the current *P* capacity to obtain the residual capacity, and its complexity is O (P-|M|); thus, the module complexity is O (EK + (P-|M|)).

Backup Reliable Path routing: it's an enabler of the best path among *k* paths to route the affected flows, it examines *n* number of Paths (P) to choose the P which meets the design objective. Its complexity depends on the number of *k* paths in *n* paths and the length of *P* in *K*. The complexity for selecting *k* out of *n* paths for one destination is O(*nk*).

4 Experimental and Performance Evaluation

The simulation is run on a machine with 3.60 GHz and 16 GB Random Access Memory (RAM) equipped with a Ryu SDN controller and Mininet version 2.2.2 [41]. Although several SDN controllers exist, Ryu is deployed as it supports all the versions of OpenFlow [42]. The simulation was conducted with Spain's backbone network topology [43]. The topology is created by writing a script in the Mininet emulator. Each switch is deployed using a software switch (OpenVswitch) in Mininet running on virtual machines. Typically, the OpenFlow switch is constrained with a limited capacity [44], and OpenVswitch can support many entries. The study modifies OpenVswitch to support the limited as in the actual OpenFlow switch. The link bandwidth connecting switches is set to 50 mbps because it gives better delay, as used and verified in [45]. The study also assumed traffic distributions follow Poison models. D-ITG is used to generate the traffic based on the Poison model with different packet sizes and duration. Random traffic flows demand pairs corresponding to a source and destination in the network were generated in Mininet. Iperf and Wireshark utility was used to analyze the throughput and packet losses.

4.1 Result and Discussion

In this section, the simulation results are reported, and the performance of the proposed scheme is evaluated against its counterpart benchmarking work under different scenarios. The results of the

proposed solution are compared on the following metrics: packet loss, throughput, and round-trip time. It is effectiveness and improvement were also discussed and analyzed in the following subsections.

4.1.1 Path Load

Path load refers to the summation of the transmission rate (Tx) and receiving rate (Rx) in the switch port based on the selected path, derived from $P_{load} = Tx_{Sp} + Rx_{Sp}$ [46]. Traffic flows were generated with different packet size variations and rates ranging from different durations based on Eq. (14). considering the residual capacity of the selected path. Fig. 5 shows the behaviour of RFRC before and after the occurrences of link failure. This is to verify the effectiveness of the proposed solution before comparing it with other benchmarking solutions. Before the failure state, flows were generated using a static link state. As such, all flows tend to compete for the shortest path. Traffic flows were transmitted accordingly. In this situation, failures were introduced on the links of the selected path at different time intervals. Therefore, RFRC reroutes the affected flows once the failure occurs based on the dynamic link state. As such, flows are redirected to a path with sufficient residual capacity. Fig. 6 compared the path load with benchmarking RPF to further verify the congestion probability's effectiveness.

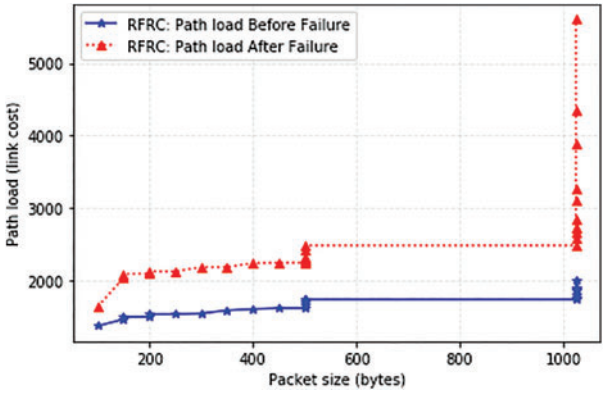

Figure 5: Path load

As shown in Fig. 6, RFRC can effectively predict the path with enough residual capacity to accommodate the number of affected flows while the baseline lacks such features, and therefore path load increases. The baseline work redirected the affected flows through any available path. Since flows vary, many flows containing large packets constantly follow the same path. As a result, the path load increases. It is effective to conclude that RFRC efficiently balanced the load on the path; therefore, congestion will be minimal with packet loss.

4.1.2 Packet Loss

Average packet loss is the number of lost data packets between the failure state and the recovery time between the source and destination. Fig. 7 shows the performance comparison of RFRC, Rapid Restoration Technique (RRT), and Fast Reliable Technique (FRT). The x-axis and y-axis represent the average packet loss for different time intervals. After the failure, it was observed that the failure detection time of RRT and FRT was slightly longer than RFRC.

Due to the detection mechanism applied. The two benchmarking works used an event-based while RFRC used periodically. The periodic approach can cope with frequent topological changes; therefore, a path with sufficient residual capacity is obtained. Incorporating the Bayesian approach is

another plus for RFRC because it regularly obtains and reroutes the affected flows through a path with sufficient capacity. This way, the number of losses was reduced to 87% as compared to RRT and FRT, respectively. This indicates that the affected flows are redirected to better link quality with minimal congestion under different traffic patterns. In comparison, the counterparts overlooked such features. Moreover, RFRC ensures that the selected alternative paths have minimal correlation or sharing links between primary and backup paths. In addition, RFRC removes only the affected rules on the failed link, this further speeds up the processing time to restore the new path, which contributes to reducing the number of packet losses.

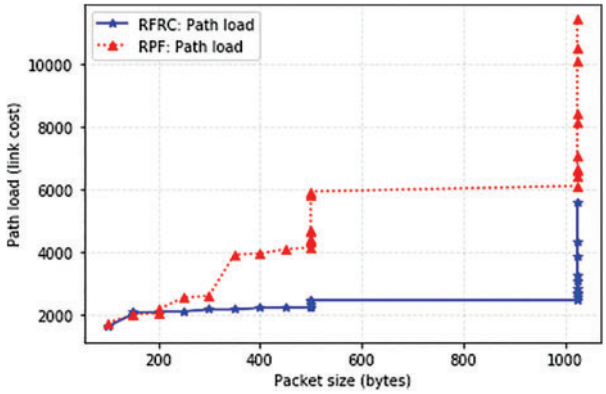

Figure 6: Comparison of RFRC and RPF path load

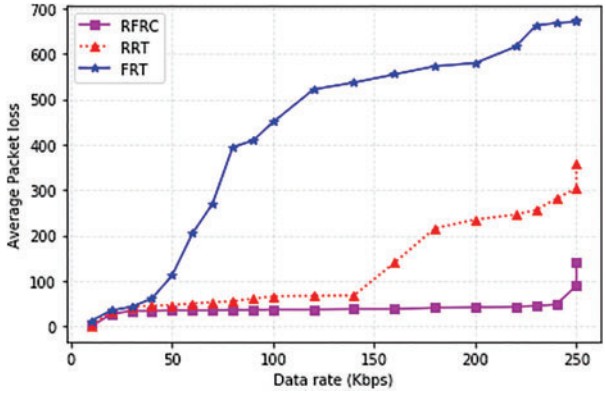

Figure 7: Average packet loss

4.1.3 Round Trip Time (RTT)

RTT is the time it takes for packets to get from source to destination [47]. RTT increases during link failure situation, which in turn affect the system throughput. Fig. 8 presents the RTT comparison between the three solutions. The average RTT of RFRC is lower than RRT and FRT with respect to the different data rates. This is because RFRC provides more timely detection of share links and affected flows are not forward through a critical path. Moreover, the set of switches on the selected has faster forwarding rules updating time. Besides, the removal of forwarding rules on the link connecting the affected path has significantly lowered the processing, decreasing the RTT for RFRC. Thus, RFRC outperforms the benchmarking works on average, it lower RTT by 89%.

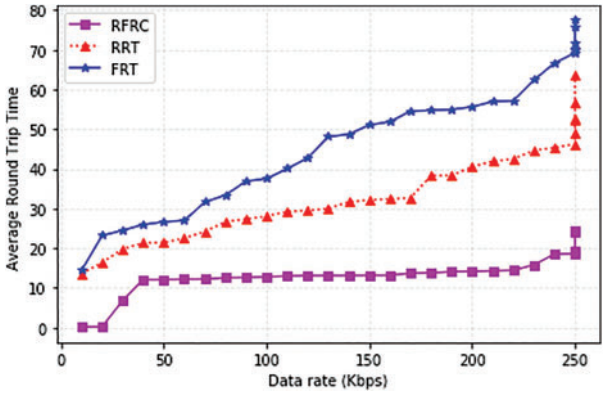

Figure 8: Average round trip time

4.1.4 Throughput

Throughput is the average amount of data successfully transferred between a source and a destination host over a communication link. Network throughput is affected by higher loss and RTT, especially during failure, before establishing an alternative path to reroute the affected flows.

Therefore, Figs. 9 and 4 translate the throughput likelihood for each scheme. Because throughput is significantly affected when the number of packet losses is higher, similarly high RTT contributes to a decline in throughput. Moreover, the degree of network congestion further augments, bringing the system throughput down. Not surprisingly, it can be observed from Fig. 5 show the throughput of the three schemes. The acquired throughput using various data rates shows that the proposed RFRC scheme performs better than benchmarking works RRT and FRT. The RFRC scheme achieved 45.96% and 45.24% improved throughput performance as compared to RRT and FRT, respectively. The variation in the curve comes as a result of network load variation. RFRC frequently examine the path load before rerouting the affected flows. In comparison, the benchmarking papers overlooked residual path load before enabling the backup path. The behaviour of the proposed scheme still maintained high throughput even when the traffic ejection increased with different data rates. Critical path avoidance also contributes because frequently used paths are avoided to reduce further congestion, identified as one factor that declined throughput. These factors are overlooked in the counterpart work and result in lower throughput.

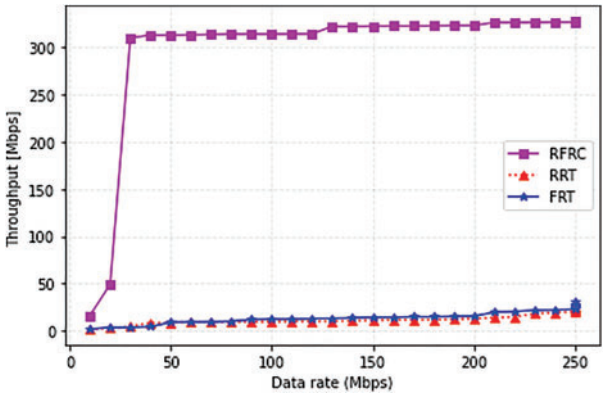

Figure 9: Average throughput

5 Conclusion

This research work presents RFRC aimed to address an established reliable backup path with post-recovery congestion awareness after occurrences of failure. The RFRC consists of two main phases: critical path detection and post-recovery congestion prediction. In critical path detection, important links are detected and assigned a score. Afterwards, the upper and lower bounds of links important to making the path are estimated. Each path is appended with its important score, a path with higher importance indicates a higher potential for future failure. Such paths are avoided to route the affected flows upon occurrences of failure. Conversely, shared links between primary and backup are examined, and the potential post-recovery congestion is predicted using Bayesian probability. The path with higher congestion probability is avoided, and the affected flows are redirected through a path with minimal post-recovery congestion. The simulated result verified the performance of RFR improved throughput by (45%), reduced packet losses (87%), and lower round-trip time (89%) compared to benchmarking works.

Acknowledgement: The authors appreciate UTM and the Deanship of Scientific Research at King Khalid University for supporting this research work under grant No R.J130000.7709.4J561 and RGP.2/111/43, respectively. In addition, one of the authors would like to thank Sule Lamido University Kafin Hausa, Nigeria, for allowing him to serve as a Researcher at Universiti Teknologi Malaysia under the Post-Doctoral Fellowship Scheme with grant No Q.J130000.21A2.06E03.

Funding Statement: The authors thank the UTM and Deanship of Scientific Research at King Khalid University for funding this work through grant No R.J130000.7709.4J561 and Large Groups. (Project under grant number (RGP.2/111/43)).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] P. Tam, S. Math and S. Kim, "Optimized multi-service tasks offloading for federated learning in edge virtualization," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 1, pp. 1–17, 2022.

[2] N. A. El-Hefnawy, O. A. Raouf and H. Askr, "Dynamic routing optimization algorithm for software defined networking," *Computers, Materials and Continua*, vol. 70, no. 1, pp. 1349–1362, 2021.

[3] B. Isyaku, M. Soperi, M. Zahid, M. B. Kamat, K. A. Bakar *et al.,* "A Ghaleb Software defined networking flowtable management of OpenFlow switches performance and security challenges: A survey," *Future Internet*, vol. 12, no. 9, pp. 147, 2020.

[4] P. M. Mohan, T. Truong-huu and M. Gurusamy, "Fault tolerance in TCAM-limited software defined networks," *Computer Networks*, vol. 116, no. 1, pp. 47–62, 2017.

[5] M. Shojaee and M. Neves, "SafeGuard: Congestion and memory-aware failure recovery in SD-WAN," in *Proc. CNSM*, Izmir, Turkey, pp. 1–7, 2020.

[6] B. Yan, Y. Xu and H. J. Chao, "Adaptive wildcard rule cache management for Software-Defined Networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 2, pp. 962–975, 2018.

[7] A. Malik, B. Aziz and M. Bader-el-den, "Finding most reliable paths for Software Defined Networks," in *Proc. IWCMC*, Valencia, Spain, pp. 1309–1314, 2017.

[8] S. Zhang, Y. Wang, Q. He, J. Yu and S. Guo, "Backup-resource based failure recovery approach in SDN data plane," in *Proc. APNOMS*, Kanazawa, Japan, pp. 1–6, 2016.

[9] S. Feng, Y. Wang, X. Zhong, J. Zong, X. Qiu *et al.,* "A ring-based single-link failure recovery approach in SDN data plane," in *Proc. NOMS*, Taipei, Taiwan, pp. 1–7, 2018.

[10] D. Adami, S. Giordano, M. Pagano and N. Santinelli, "Class—based traffic recovery with load balancing in SDN," in *Proc. GC Wkshps*, Austin, TX, USA, pp. 161–165, 2014.

[11] Y. Wang, S. Feng and H. Guo, "A single-link failure recovery approach based on resource sharing and performance prediction in SDN," *IEEE Access*, vol. 7, no. 1, pp. 174750–174763, 2019.

[12] G. Iannaccone, C. Chuah, R. Mortier, S. Bhattacharyya and C. Diot, "Analysis of link failures in an IP backbone," in *Proc. 2nd ACM SIGCOMM*, Marseille, France, pp. 237–242, 2002.

[13] A. Malik and R. De Fréin, "Rapid restoration techniques for Software-Defined Networks," *Applied Sciences*, vol. 10, no. 10, pp. 1–24, 2020.

[14] Z. Zhu, Q. Li, S. Xia and M. Xu, "CAFFE: Congestion-aware fast failure recovery in Software Defined Networks," in *Proc. ICCCN*, Hangzhou, China, pp. 1–9, 2018.

[15] Q. Li, Y. Liu, Z. Zhu, H. Li and Y. Jiang, "BOND: Flexible failure recovery in software defined networks," *Computer Networks*, vol. 149, no. 5, pp. 1–12, 2019.

[16] Z. Cheng, X. Zhang, Y. Li, S. Yu, R. Lin *et al.,* "Congestion-aware local reroute for fast failure recovery in Software-Defined Networks," *Journal of Optical Communications and Networking*, vol. 9, no. 11, pp. 934, 2017.

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

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

[19] A. Markopoulou, G. Iannaccone and S. Bhattacharyya, "Characterization of failures in an operational IP backbone network," *IEEE/ACM Transactions on Networking*, vol. 16, no. 4, pp. 749–762, 2008.

[20] S. Sharma, D. Staessens, D. Colle, M. Pickavet and P. Demeester, "Enabling fast failure recovery in OpenFlow networks," in *Proc. DRCN*, Krakow, Poland, pp. 164–171, 2011.

[21] S. Sharma, D. Staessens, D. Colle, M. Pickavet and P. Demeester, "OpenFlow: Meeting carrier-grade recovery requirements OpenFlow," *Computer Communications*, vol. 36, no. 6, pp. 656–665, 2013.

[22] B. Isyaku, K. A. Bakar, M. S. M. Zahid, E. H. Alkhammash, F. Saeed *et al.,* "Route path selection optimization scheme based link quality estimation and critical switch awareness for Software Defined Networks," *Applied Sciences*, vol. 11, no. 19, pp. 9100, 2021.

[23] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci and P. Castoldi, "OpenFlow-based segment protection in ethernet networks," *Journal of Optical Communications and Networking*, vol. 5, no. 9, pp. 1066–1075, 2013.

[24] J. Kempf, E. Bellagamba, A. Kern and A. Ab, "Scalable fault management for OpenFlow," in *Proc. ICC*, Ottawa, ON, Canadam, pp. 6606–6610, 2012.

[25] H. Kim, J. R. Santos, Y. Turner, M. Schlansker, J. Tourrilhes *et al.,* "CORONET: Fault tolerance for Software Defined Networks," in *Proc. ICNP*, Austin, TX, USA, pp. 1–2, 2012.

[26] S. Astaneh and S. S. Heydari, "Multi-failure restoration with minimal flow operations in Software Defined Networks," in *Proc. DRCN*, Kansas City, MO, USA, pp. 263–266, 2021.

[27] B. Isyaku, M. S. M. Zahid, M. B. Kamat, K. A. Bakar and F. A. Ghaleb, "Software defined networking failure recovery with flow table aware and flows classification," in *Proc. ISCAIE*, Penang, Malaysia, pp. 337–342, 2021.

[28] S. Hegde, "Path restoration in source routed Software Defined Networks," in *Proc. ICUFN*, Milan, Italy, pp. 720–725, 2017.

[29] A. L. I. Malik, B. Aziz, C. Ke, H. A. N. Liu and M. O. Adda, "Virtual topology partitioning towards an efficient failure recovery of Software Defined Networks," in *Proc. ICMLC*, Ningbo, China, pp. 646–651, 2020.

[30] A. L. I. Malik, B. Aziz, M. O. Adda and C. Ke, "Optimisation methods for fast restoration of Software-Defined Networks," *IEEE Access*, vol. 5, no. 1, pp. 16111–16123, 2017.

[31] K. Gotani, H. Takahira, M. Hata, L. Guillen and S. Izumi, "Design of an SDN control method considering the path switching time under disaster situations," in *Proc. ICT-DM*, Sendai, Japan, pp. 18–21, 2018.

[32] K. Gotani, H. Takahira, M. Hata, L. Guillen, S. Izumi *et al.,* "OpenFlow based information flow control considering route switching cost," in *Proc. COMPSAC*, Milwaukee, WI, USA, pp. 2019–2022, 2019.

[33] P. M. Mohan, T. Truong-huu and M. Gurusamy, "TCAM-aware local rerouting for fast and efficient failure recovery in Software Defined Networks," in *Proc. GLOBECOM*, San Diego, CA, USA, pp. 1–6, 2015.

[34] J. Chen, J. Chen, J. Ling and W. Zhang, "Failure recovery using Vlan-tag in SDN: High speed with low memory requirement," in *Prco. IPCCC*, Las Vegas, NV, USA, pp. 1–9, 2016.

[35] H. Liaoruo, S. Qingguo and S. Wenjuan, "A source routing based link protection method for link failure in SDN," in *Proc. ICCC*, Chengdu, China, pp. 2588–2594, 2016.

[36] P. Thorat, R. Challa, S. M. Raza, D. S. Kim and H. Choo, "Proactive failure recovery scheme for data traffic in SDN," in *Proc. NetSoft*, Seoul, Korea (South), pp. 219–225, 2016.

[37] 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 *Proc. ICC*, Kuala Lumpur, Malaysia, pp. 1–6, 2016.

[38] B. Stephens, A. L. Cox and S. Rixner, "Scalable multi-failure fast failover via forwarding table compression," in *Proc. Symp. on SDN Research*, Santa Clara, CA, US, no. 9, pp. 1–12, 2016.

[39] P. Thorat, S. M. Raza, D. S. Kim and H. Choo, "Rapid recovery from link failures in Software-Defined Networks," *Journal of Communications and Networks*, vol. 19, no. 6, pp. 648–665, 2017.

[40] P. Thorat, S. Jeon, S. M. Raza and H. Choo, "Pre-provisioning of local protection for handling Dual-failures in OpenFlow-based Networks," in *Proc. CNSM*, Tokyo, Japan, pp. 1–6, 2017.

[41] B. Lantz, B. Heller and N. Mckeown, "A network in a laptop: Rapid prototyping for Software-Defined Networks," in *Proc. 9th ACM SIGCOMM Workshop*, Monterey, CA, USA, pp. 1–6, 2010.

[42] L. Zhu, K. Sharif, F. Li, X. Du and M. Guizani, "SDN controllers: Benchmarking & performance evaluation," *IEEE JSAC*, vol. 19, no. 1, pp. 1–14m, 2019.

[43] Z. Guo, R. Liu, Y. Xu, A. Gushchin, A. Walid *et al.,* "STAR: Preventing flow-table overflow in Software-Defined Networks," *Computer Networks*, vol. 125, no. 1, pp. 15–25, 2017.

[44] R. Li and X. Wang, "A Tale of two (flow) tables: Demystifying rule caching in OpenFlow switches," in *Proc. CPP*, Kyoto, Japan, pp. 1–10, 2019.

[45] L. Huang, X. Zhi, Q. Gao, S. Kausar and S. Zheng, "Design and implementation of multicast routing system over SDN and sFlow," in *Proc. ICCSN*, Beijing, China, pp. 524–529, 2016.

[46] M. F. Rangkuty and M. H. A. Al-hooti, "Path selection in Software Defined Network data plane using least loaded path," in *Proc. ICACSIS*, Depok, Indonesia, pp. 135–140, 2020.

[47] U. Zakia and H. Ben Yedder, "Dynamic load balancing in SDN-based data centre networks," in *Proc. IEMCON*, Vancouver, BC, Canada, pp. 242–247, 2017.