Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2024.054215

ARTICLE

Enhanced Mechanism for Link Failure Rerouting in Software-Defined Exchange Point Networks

Abdijalil Abdullahi[1,2] and Selvakumar Manickam[2,*]

[1]Department of Information Technology, SIMAD University, Mogadishu, 630, Somalia

[2]National Advanced IPv6 Centre, Universiti Sains Malaysia, Pulua Pinang, 11800, Malaysia

*Corresponding Author: Selvakumar Manickam. Email: selva@usm.my

Received: 22 May 2024 Accepted: 09 August 2024 Published: 12 September 2024

ABSTRACT

Internet Exchange Point (IXP) is a system that increases network bandwidth performance. Internet exchange points facilitate interconnection among network providers, including Internet Service Providers (ISPs) and Content Delivery Providers (CDNs). To improve service management, Internet exchange point providers have adopted the Software Defined Network (SDN) paradigm. This implementation is known as a Software-Defined Exchange Point (SDX). It improves network providers' operations and management. However, performance issues still exist, particularly with multi-hop topologies. These issues include switch memory costs, packet processing latency, and link failure recovery delays. The paper proposes Enhanced Link Failure Rerouting (ELFR), an improved mechanism for rerouting link failures in software-defined exchange point networks. The proposed mechanism aims to minimize packet processing time for fast link failure recovery and enhance path calculation efficiency while reducing switch storage overhead by exploiting the Programming Protocol-independent Packet Processors (P4) features. The paper presents the proposed mechanisms' efficiency by utilizing advanced algorithms and demonstrating improved performance in packet processing speed, path calculation effectiveness, and switch storage management compared to current mechanisms. The proposed mechanism shows significant improvements, leading to a 37.5% decrease in Recovery Time (RT) and a 33.33% decrease in both Calculation Time (CT) and Computational Overhead (CO) when compared to current mechanisms. The study highlights the effectiveness and resource efficiency of the proposed mechanism in effectively resolving crucial issues in multi-hop software-defined exchange point networks.

KEYWORDS

Link failure recovery; Internet exchange point; software-defined exchange point; software-defined network; multi-hop topologies

1 Introduction

Since its start in the early 1990s, the Internet has experienced substantial changes. At first, it was a cutting-edge spreading infrastructure with a tiny number of users, mainly managed by a small set of Tier-1 transit providers in charge of worldwide connections [1,2]. The hierarchical network structure,

primarily dependent on major transit providers, faced challenges due to changing network traffic needs and interconnection requirements [3].

This eventually shifted towards a flatter topology supported by direct peering connections [4]. This development introduced Internet Exchange Points (IXPs) as crucial components in the structure of the Internet, with a focus on enhancing network bandwidth and lowering interconnection expenses [5,6]. Internet exchange points have been crucial in the Internet peering ecosystem during the last few decades, significantly boosting the global participation of providers and promoting the development of peering partnerships [7]. Traditional Internet exchange point architecture has faced limitations, especially in managing complex network dynamics like broadcast storms and the inefficiencies in using the Border Gateway Protocol (BGP) for routing decisions [8,9].

The advent of Software-Defined Networking (SDN) has offered a revolutionary approach to network management [10,11]. It is characterized by the separation of the control and data planes, provides logically centralized control of programmable switches, facilitating traffic management at a more granular level compared to the coarse-grained level routing of Border Gateway Protocol (BGP) [12,13]. Building upon the principles of SDN, Software-Defined Exchange Points (SDXs) emerged, offering network providers more refined control over traffic forwarding and policy implementation [14,15]. As illustrated in Fig. 1, software-defined exchange points consist of a software-defined network controller, a border gateway protocol route server, and a programmable switching fabric. These points enable more sophisticated traffic management and policy enforcement than traditional Internet exchange points.

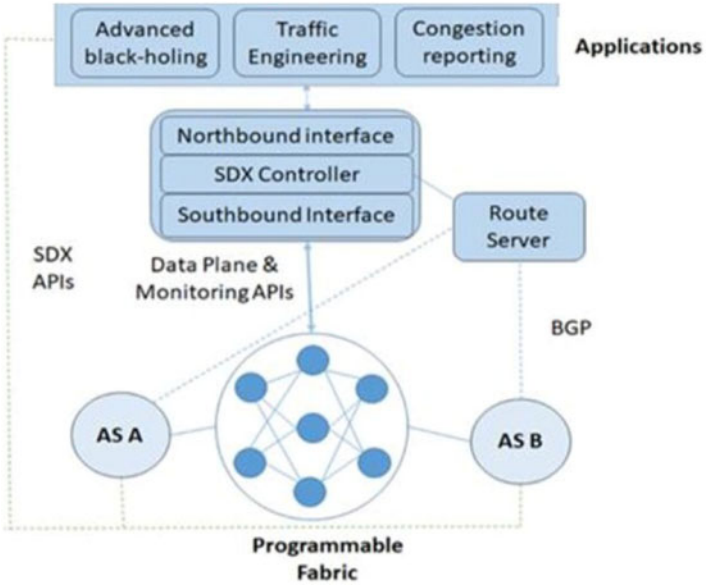

Figure 1: Software-defined exchange point architecture (Reprinted/adapted with permission from Reference [16]. Copyright 2023, Institute of Advanced Engineering and Science)

Problem Formulation Despite advancements in Software-Defined Networking (SDN) and the deployment of Software-Defined Exchanges (SDXs), there remain persistent challenges, particularly in managing link failure recovery and minimizing packet processing delays in multi-hop network configurations in software-defined exchange point environments [17,18]. The complex architecture characterized by multiple interconnected switches elevates the difficulties of routing traffic promptly

and efficiently during link disruptions [16,19]. Existing approaches to addressing these challenges have often resulted in trade-offs such as increased packet processing times, higher switch memory overheads, and lack of dynamic path computation [20,21]. These issues stem from the inherent limitations of current software-defined exchange point frameworks, which struggle to balance quick failover responses with minimal impact on network performance [22,23].

The core problem addressed in this research is the need for a more resilient and efficient mechanism for link failure recovery in software-defined exchange point environments, especially within multi-hop topologies. The proposed Enhanced Link Failure Rerouting (ELFR) mechanism aims to fundamentally improve the reliability and efficiency of these networks by optimizing link failure rerouting processes, reducing packet processing delays, and managing switch memory overhead more effectively. Through advanced algorithms and the innovative use of Programming Protocol-independent Packet Processors (P4), the proposed mechanism seeks to improve the performance and scalability of the software-defined exchange point environments. The contributions of this paper can be summarized as follows:

- This research paper introduces Enhanced Link Failure Rerouting (ELFR), a novel mechanism to quickly recover from link failures in software-defined exchange point networks, focusing on multi-hop architectures. The proposed mechanism comprises two key modules: packet processing and path computation. The packet processing module quickly and accurately manages data packets to minimize disruptions during failures, while the path computation module efficiently devises alternative routes to maintain data flow. These integrated modules improve software-defined exchange point networks' robustness, resilience, and performance, ensuring reliable data transmission for complex tasks.
- The proposed mechanism leverages the capabilities of protocol-independent packet processors (P4) to minimize recovery times during a link failure swiftly. It dynamically adds custom data to packet headers, distinguishing normal packets from recovery packets—'0' for normal and '1' for those requiring rerouting. This classification lets switches insert and read backup path data from packet headers, enabling them to redirect affected packets along alternate routes swiftly. This efficient process significantly simplifies the packet forwarding and reduces downtime when link failures occur.
- The proposed mechanism enhances software-defined exchange point networks by quickly calculating backup paths using the software-defined network controller's advanced path computation strategy. This strategy incorporates Programming Protocol-independent Packet Processors (P4) functionalities to determine and store neighbor-specific backup routes proactively. This approach optimizes storage and computational resources by having each switch hold only directly connected neighbor backup data. The proposed mechanism efficiently computes and installs the shortest backup paths between switches through an optimized algorithm, ensuring rapid and effective rerouting during link failures.
- In our study, we evaluated the proposed mechanism, focusing on recovery time, calculation time, and memory overhead. Results show that the proposed mechanism significantly outperforms the ENDEAVOUR mechanism, with lower metrics in all categories—37.5% in recovery time and 33.33% in computation time and overhead, *vs.* ENDEAVOUR's 62.5% and 66.67%, respectively. The proposed mechanism provides better efficiency in recovery time, faster path computation, and reduced computational overhead.

Following this introduction, the subsequent sections of this paper are organized as follows: Section 2 provides related work by critically reviewing existing mechanisms. Section 3 offers the design

of the proposed mechanism. Section 4 presents the experimental results and discussion. Finally, Section 5 provides the conclusion and future work of the paper.

2 Related Work

This section describes the existing methods and frameworks for recovering from link failure. The approaches and frameworks are separated into Software-Defined Exchange Point (SDX) based and Fast Re-Routing (FRR) based frameworks. To improve link failure rerouting management, software-defined network-based frameworks are employed on Internet exchange point providers to utilize software-defined network features, notably multi-hop topologies. Fast Re-Routing (FRR) based algorithms and approaches still need to be implemented in the software-defined exchange point environment's multi-hop architecture. However, they suggest viable schemes to reroute lost packets promptly utilizing non-software-defined network methods.

2.1 Software-Defined Exchange Point-Based Frameworks

Internet exchange point providers have had issues addressing link failure rerouting, particularly in multi-hop topologies. This section covers two frameworks: ENDEAVOUR [21], and Umbrella [20], meant to solve the issues of Internet exchange points' fast failover recovery and their recommended solutions.

ENDEAVOUR is a software-defined networking framework created for Internet exchange point operators and implemented on multi-hop Internet exchange point topologies. It decreases switching fabric challenges caused by broadcast traffic and promotes the scalability of Internet exchange point providers. This framework provided three ways during link failure recovery in multi-hop topologies. The methods involve enforcing duplicate outgoing rules, returning packets to the ingress switch, and inserting recovery information into packets. These solutions have addressed many issues related to link failure recovery but have not fixed the latency in packet processing and the overhead on switch memory and path computation.

The Umbrella is a software-defined exchange point method that can provide a reliable and solid switching fabric, reducing the potential for controller failures. Furthermore, Umbrella can be implemented on any single-hop or multi-hop Internet exchange point topology. Umbrella leverages OpenFlow (OF) technologies to address data plane link failure. OpenFlow leverages the group's quick failover functionality to address an interrupted link. This functionality enables the monitoring of interfaces, switch port states, and forwarding action independently of a controller. Recovering the data plane after a failure is very tough. The Umbrella controller implements Bidirectional Forwarding Detection (BFD) and Local Link Discovery (LLD). The Umbrella controller updates the configuration of the edge switch with the backup route when a data plane link loss is detected. The Umbrella framework could not address the constraints of connection loss in a multi-hop topology due to the dependable rerouting capabilities of OpenFlow.

2.2 Fast Re-Routing (FRR)-Based Frameworks

This section describes the general frameworks for Fast Re-Routing (FRR) in the event of link failures, such as Software Implemented Fault Tolerance (SWIFT) [24,25]; Primitive for Reconfigurable Fast Reroute (PURR) [26], and Fast Re-Routing (FRR) [27].

Software Implemented Fault Tolerance (SWIFT) provides a quick rerouting mechanism that allows routers to recover swiftly from distant failures. The framework uses two methods: first, it

predicts distant failures by analyzing a small number of Border Gateway Protocol (BGP) updates it has received. Secondly, the framework suggested an encoding approach for the data plane that allows rapid and adaptable updates to the affected forwarding entries. The framework decreased the time needed for convergence and addressed the issues related to remote outages in transit networks. Primitive for Reconfigurable Fast Reroute (PURR) eliminates packet recirculation for minimal failover delay and excellent switch performance. It demonstrates robust resilience to numerous simultaneous failures and facilitates recovery from failures with little memory usage. This method introduced a fast-rerouting primitive for programmable data planes, enabling failover mechanisms to integrate without recirculating packets, resulting in minimal failover latency and increased switch throughput. Fast Re-Routing (FRR) introduced a rapid rerouting framework to restore router interruptions and quickly reroute traffic in case of failure. This framework changed the pre-computed routing path, allowing for rapid rerouting. It offers distinct failover routes to minimize the impact of interruptions and packet loss, improving fast re-routing protection performance.

2.3 Critical Review

This section provides a thorough critical analysis of the current frameworks. Current Internet exchange points utilize rapid rerouting by employing traditional routing technologies like Open Shortest Path First (OSPF) and Multiprotocol Label Switching (MPLS). The systems redirect packets to specific egress ports and predetermined alternate routes in case of an internal connection breakdown. Rerouting packets in a single switch topology is typically straightforward when a link failure occurs. Since a link failure affects multiple switches within the Internet exchange point, packet rerouting in a multi-hop topology becomes more difficult. Internet exchange points provide rerouting packets using OpenFlow techniques like EDEAVOUR and Umbrella for failover recovery. However, these mechanisms are ineffective in quickly rerouting packets and do not excel in packet processing performance. Conversely, several link failure rerouting strategies have been suggested, including a Software Implemented Fault Tolerance (SWIFT) [24]; Primitive for Reconfigurable Fast Reroute (PURR) [26], and Fast-Re-Routing (FRR) [27]. The techniques effectively carried out quick rerouting but were not utilized for multi-hop software-defined exchange point topologies. Therefore, these approaches can improve fast failover recovery techniques and minimize packet processing delays in multi-hop software-defined exchange point topologies.

ENDEAVOUR [21] suggested solutions that addressed specific difficulties, but there are still constraints, such as packet processing delay and switch memory overhead. Umbrella [20] proposed solutions that mitigate certain constraints, although they rely on OpenFlow elements that cause delays in connection failure recovery. Software Implemented Fault Tolerance (SWIFT) [24] introduced ways to address specific issues with quick rerouting. Still, these mechanisms faced restrictions such as rerouting unaffected prefixes, leading to potential overhead and accuracy issues. The proposed methods could not reroute a significant number of prefixes but could reroute a small amount.

A Fast Re-Routing (FRR) primitive for modifiable data planes was proposed via the Primitive for Reconfigurable Fast Reroute (PURR) [26] mechanism. Integrating the existing failover techniques without recirculating packets results in less failover latency and more switch performance. The software-defined exchange point environment covered in this work does not support implementing this function. This mechanism might improve that study's techniques. This mechanism might improve the study's approaches. Fast Re-Routing (FRR) [27] proposed a technique for single topologies in the software-defined exchange point environment; however, it has not yet been implemented in multi-hop software-defined exchange point topologies. This mechanism aims to address the issues related to link failure recovery. Additionally, the method uses two-stage forwarding rules to modify the data plane,

which delays the interruption's rerouting. For routing link failures, Table 1 highlights the mechanisms, benefits, and limitations of current frameworks and approaches, including those based on Software-Defined Exchange Points (SDX) and Fast Re-Routing (FRR).

Table 1: Analysis of current link failure rerouting mechanisms

Frameworks	Mechanisms	Strength	Drawbacks
ENDEAVOUR [21]	Replicate external policies	Duplicating all members' outgoing and incoming policies across all switches.	Substantial forwarding state duplication.
	Bounce packets back to ingress switch	Prevents unnecessary duplication of forwarding state.	Waste of bandwidth and increased packet latency.
	Insert packets with recovery information	All overheads from the previous methods were resolved.	Cost of switch memory and processing delay.
Umbrella [20]	OpenFlow group fast failover	It switches forwarding activities independently, without needing a controller and monitors the state of ports and interfaces.	Without the controller, the data plane failure cannot be recovered.
	Local link discovery protocol (LLDP) and Bidirectional forwarding detection (BFD) protocol	It resolved the problem of recovering from data plane failure, which was brought on by the above method.	It relies on the controller, which leads to a delay in recovery.
Software implemented fault tolerance (SWIFT) [24]	Inference algorithm	Determines which prefixes will have minimal outage notification. Rerouting the affected prefixes on paths unaffected by the inferred failure.	It may have rerouted unaffected prefixes, making it impossible to control both the accuracy and the speed of rerouting the impacted prefixes concurrently.

(Continued)

Table 1 (continued)

Frameworks	Mechanisms	Strength	Drawbacks
	Encoding scheme	Enables quick and flexible updates of the affected forwarding entries.	Capable of only rerouting a small number of prefixes rather than a large number of them. The first border gateway protocol update can take minutes to disseminate upon a data plane failure.
Primitive for reconfigurable fast reroute (PURR) [26]	Primitive for programmable data plane	Reduces failover latency and increases switch throughput. It can handle both single and multiple failures. Stops packets from being circulated.	Restricted to failures of the data plane. Deployment is not feasible on multi-hop Internet exchange point topologies.
Fast re-routing (FRR) [27]	Fast re-routing	Determines the cause of the routing disruption. Uses a quick rerouting method. Decreases the interruption's recovery time.	Uses two-stage forwarding rules to enable data plane updates; however, redirecting the interruption may take some time. Incapable of being deployed on multi-hop IXP topologies.
Proposed mechanism	Packet processing scheme	Reduces recovery time for packet processing during link failures in a software-defined exchange point. Minimizes the storage overhead of switch memory.	Applicable only to single-link failure scenarios. In single-link failure schemes, loop issues may occur due to slow convergence, inconsistent routing information, or improperly managed redundant paths.

(Continued)

Table 1 (continued)

Frameworks	Mechanisms	Strength	Drawbacks
	Path computation scheme	Enables dynamic path computation with minimized calculation time in a software-defined exchange point. Recovers link failures by identifying the shortest backup path.	

The proposed mechanism presents an innovative method for handling packet processing and path computation in software-defined exchange points, explicitly focusing on effectively dealing with the difficulties associated with link failure recovery. This mechanism effectively decreases the time it takes to recover from link failures compared to traditional methods. It achieves this by optimizing how packets are processed and lowering the storage space used in switch memory. Furthermore, it improves the effectiveness of path calculation by allowing for real-time calculation with decreased processing time, ensuring rapid discovery of the shortest alternative path.

3 Proposed Mechanism

This section proposes an Enhanced Link Failure Rerouting (ELFR) mechanism to solve the issues raised in the critical review section. The proposed method decreases packet processing latency, enhances path computation, and lowers memory storage overhead to improve the performance of Software-Defined Exchange Points (SDX), especially link failure recovery for multi-hop topologies. The subsequent subsections outline the requirements of the mechanism, architecture, and modules for the proposed mechanism.

3.1 Requirement of Proposed Mechanism

The primary objectives of this research are to validate the proposed mechanism's output and accomplish its main goal. The proposed mechanism must meet the following needs:

- The proposed mechanism should be built on the Software-Defined Exchange Point (SDX) using Programming Protocol-independent Packet Processors (P4) capabilities.
- When a link fails, the proposed mechanism must prevent packet recirculation, which raises packet processing latency.
- The proposed mechanism should not treat any normal forwarding packet as a recovery packet; doing so would delay recovery.
- The proposed mechanism should not compute the recovery backup path after the link fails.

3.2 Proposed Mechanism Architecture

This section outlines the proposed mechanism's architecture. The primary objective of this proposed mechanism is to reduce link failure recovery delays in multi-hop software-defined exchange

points. In the software-defined exchange point multi-hop architecture, the proposed mechanism enhances path calculation to compute a recovery backup path effectively and reduce packet processing latency in case of a link failure. Fig. 2 shows the proposed architecture of the proposed mechanism. The proposed mechanism is comprised of two distinct modules, each incorporating its components within the architectural mechanism:

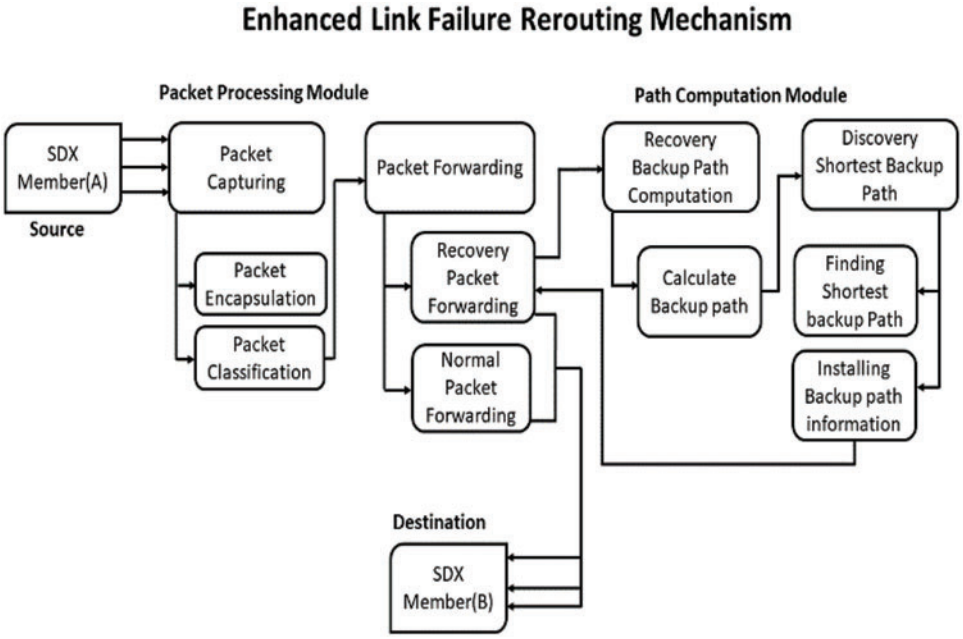

Figure 2: Architecture of proposed mechanism (Reprinted/adapted with permission from Reference [16]. Copyright 2023, Institute of Advanced Engineering and Science)

3.2.1 Packet Processing Module

After packets are encapsulated and designated, this module aims to route them to their destination efficiently. It lets us distinguish recovery packets from normal packets. A packet affected by a link failure needs to recover quickly and efficiently. The module will process and normally forward the packet if it seems normal.

3.2.2 Path Computation Module

This module lets the user choose the shortest backup path if a link fails proactively. It also uses a neighbor-based technique to deploy the shortest recovery path on switches.

3.3 Proposed Mechanism

In this section, the proposed method is explained. The packet processing and path computation modules compose the proposed method. Modules consist of various processes. Packet forwarding, packet capture for packet processing, recovery backup path computation, and shortest backups for the path computation module are some of the processes involved. Key steps within each process include packet encapsulation, packet classification, usual and recovery packet forwarding, backup

path computation, backup path shortest path finding, and backup path information installation. The proposed mechanism's components and details are listed below.

3.3.1 Packet Processing Module

This module utilizes Software-Defined Networking (SDN) methods and Programming Protocol-independent Packet Processors (P4) to enhance packet processing and forward the packets effectively in case of link failure. Link failure recovery in Software-Defined Networking (SDN) is a crucial issue often addressed via proactive or reactive methods [28,29]. This study avoids the reactive method due to recovery delays and instead implements the proactive method. Implementing a proactive recovery method promptly creates backup routes to facilitate quick traffic redirection in case of a link failure. The pre-determined routes are saved as flow entries in intermediary switches. This proactive approach can reliably meet stringent recovery time requirements, often 50 ms for carrier-grade networks. However, this speed comes at a resource utilization cost, creating challenges like flow table storage space bottlenecks and backup path performance. This research implements a proactive recovery method using P4 features to address the switch's storage overhead.

Programming Protocol-independent Packet Processors (P4) is a high-level programming language designed for protocol-independent packet processors and provides flexibility and control over packet processing [30,31]. The language enables not just the definition of packet header fields but also the customization of how these packets are parsed and processed through various stages. When a packet enters the switch, it undergoes parsing to translate it into a processable format. Following this, ingress and egress 'match–action' tables are applied to the packet to modify its headers or determine its egress port. Finally, before forwarding, the packet is deparsed based on its current state [32,33]. Integrating Programming Protocol-independent Packet Processors (P4) into software-defined network recovery strategies offers intriguing possibilities. In proactive recovery, Programming Protocol-independent Packet Processors (P4) can be leveraged to optimize the storage and application of backup paths. For example, the ingress 'match–action' tables can be programmed to implement the predefined recovery strategies, thus reducing the need for storing excessive flow entries. This can make the system more efficient while still maintaining rapid recovery times. By weaving the capabilities of Programming Protocol-independent Packet Processors (P4) together with the existing proactive recovery method in a software-defined network, we can aim to optimize this strategy and potentially revolutionize it. This fusion can provide a more robust, efficient, and agile approach for managing link failures in software-defined networks.

Using OpenFlow capabilities, ENDEAVOUR [21] and Umbrella [20], presented methods that increased switch memory costs and packet processing delays. In the software-defined network architecture, two methods for recovering from link failures are proactive and reactive. The proactive approach requires more storage, increasing the switch memory cost, whereas the reactive option adds latency to the failed recovery process. The proposed mechanism minimizes switch memory overhead and packet processing delays by employing a proactive link failure recovery technique and leveraging Programming Protocol-independent Packet Processors (P4) features [34,35]. The software-defined exchange point environment's multi-hop architecture has previously hindered the effective and efficient processing of packets using Programming Protocol-independent Packet Processors (P4) features in earlier studies. The two processes that comprise the packet processing module are packet forwarding and packet capture, which effectively process and forward packets:

- Packet Capturing: The two packet capturing steps are classification and encapsulation. Packet encapsulation uses Programming Protocol-independent Packet Processors (P4) to add custom

packets to a packet header in case of a link failure. With packet classification, you can handle packets quickly and effectively without forwarding the unaffected packet as a recovery packet. It also allows you to verify the state of the packets and distinguish between normal and recovery packets. This step separates the affected packet from the normal packet using a one-bit status field to identify the packets. The packet in link failure recovery has a **state** of **1**, while a normal packet has a **state** of **0**. If a packet is in the link failure recovery state, the backup path information must be added to the header. Thus, this process aims to collect, encapsulate, and classify the packets.

- Packet Forwarding: Normal Packet Forwarding and Recovery Packet Forwarding are the two stages of packet forwarding. Considering the preceding phase that distinguished between the two types of packets, this process manages packet forwarding. First, a normal packet must be processed normally and should not be impacted by the failure. Secondly, the failure is affected by a recovery packet, which needs to recover quickly. Normal packets are handled by Normal Packet Forwarding, which routes packets based on the destination address. On the other hand, Recovery Packet Forwarding handles recovery packets using backup path information in the packet header rather than the normal packet routing procedure. In this case, in the event of a packet link failure, the switch inserts the backup path data into the custom packet header. Subsequent switches can complete the forwarding process by retrieving the backup path data from the custom packet header. The proposed mechanism separates packets into parts to process them. First, a normal packet forwards to the destination normally using the destination address. This packet does not affect the failure, so it forwards directly. Second, the recovery packet forwards using an alternative backup path that computes the software-defined network controller. This packet does not process directly into the destination because it causes failure, as shown in Fig. 3.

Algorithm 1 shows the packet processing. The **parseHeader** function in the algorithm initiates by parsing the packet's header to determine if it's in "NORMAL" or "RECOVERY" mode. A "NORMAL" header matches the destination address against the forward table to identify the correct port, using **checkState** to assess the port's condition. If the port is active, the packet is forwarded; if inactive, the header switches to "RECOVERY," and a backup path is selected using **bpTable**. If the header is already in "RECOVERY" mode, **getForwardPort** retrieves the next port, which is validated for activity. If active, the packet is sent through this port, resetting the header to "NORMAL" if the path length is zero; otherwise, a backup path is chosen, and the process restarts. This loop continues until an active port enables successful packet forwarding.

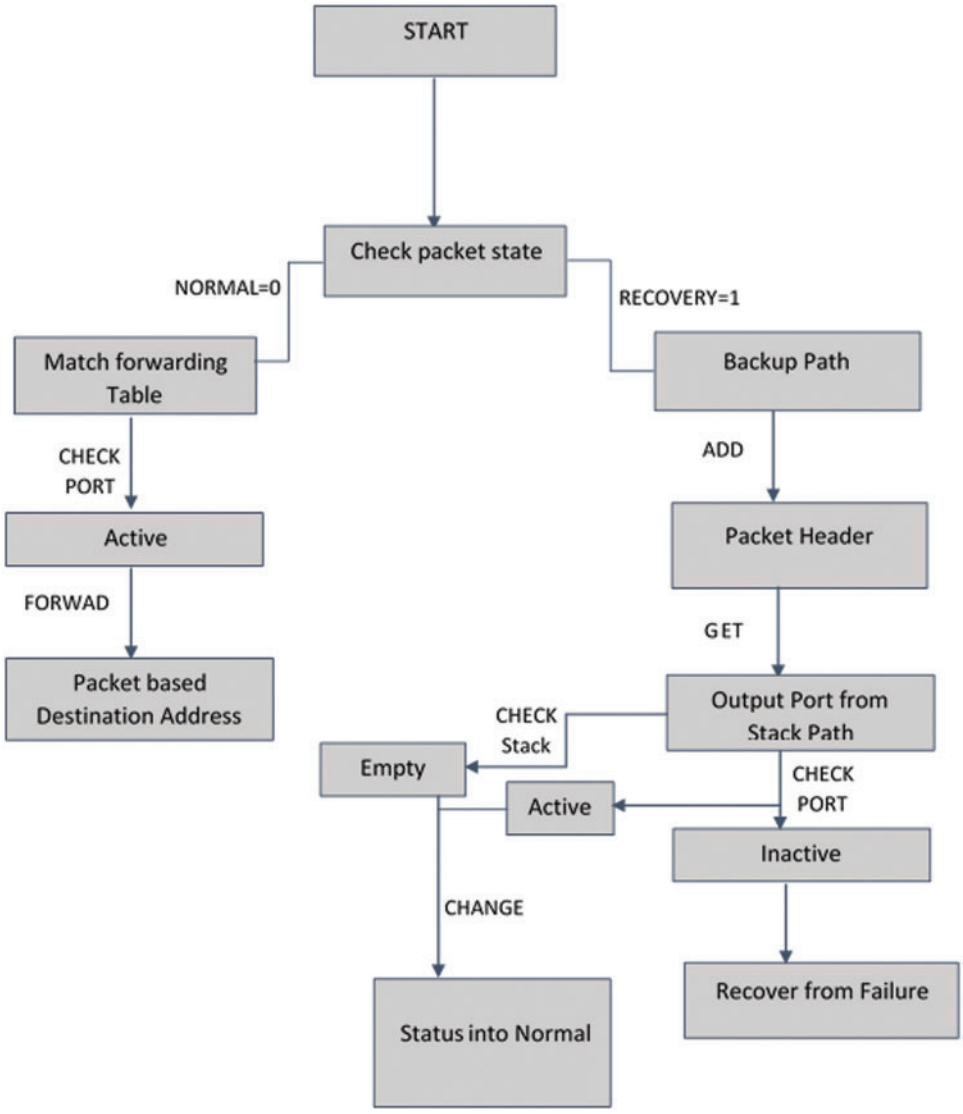

Figure 3: Workflow of packet processing (Reprinted/adapted with permission from Reference [16]. Copyright 2023, Institute of Advanced Engineering and Science)

Algorithm 1: Packet Processing Algorithm

Input: Network Packet P
Output: Port used to forward P: port
1: initial_header ← parseHeader(P)
2: if initial_header.state == NORMAL then
3: port ← matchForwardTable(initial_header.dst_address)
4: port_state ← checkState(port)
5: if port_state == active then

(Continued)

Algorithm 1 (continued)

```
6:              return port
7:        else
8:              initial_header.state ← RECOVERY
9:              bpPath ← bpTable(initial_header.dst_address)
10:             initial_header.path ← bpPath
11:             go to line 1
12:       end if
13: else
14:       port ← getForwardPort(initial_header.path.pop())
15:       port_state ← checkState(port)
16:       if port_state == active then
17:             if len(initial_header.path) == 0 then
18:                   initial_header.state ← NORMAL
19:             end if
20:             return port
21:       else
22:             bpPath ← bpTable(initial_header.dst_address)
23:             initial_header.path ← bpPath
24:             go to line 1
25:       end if
26: end if
27: final
```

3.3.2 Path Computation Module

This module aims to install the system on switches, calculate the recovery backup path before time in case of a link failure, and determine the quickest backup path. The traditional proactive failure recovery method controller must install each switch with the backup path and compute it using a destination-based approach. This module has a proactive link failure recovery mechanism that uses the neighbor-based backup path calculation method to preserve switch storage, compared to the traditional destination-based backup path calculation approach. The path computation module consists of two processes to determine the shortest backup path: the Discovery of the Shortest Backup Path and the Recovery Backup Path Computation.

- Recovery Backup Path Computation: Recovery Backup Path Calculation calculates the backup path before link failure occurs. In the software-defined network architecture, the controller determines the recovery backup path using proactive and reactive mechanisms. When a link failure is discovered during reactive recovery, the mechanism computes the backup path after detecting the failure that delays recovery. Proactive recovery involves calculating the backup path before link failure occurs, which causes storage overhead but reduces the delay of the recovery process. Leveraging Programming Protocol-independent Packet Processors (P4) capabilities, this process combines a neighbor-based backup path computation technique with a traditional proactive recovery mechanism. Each switch can store only the backup paths of the connected switches by using neighbor-based backup path calculation.
- Discovery of the Shortest Backup Path: Installing Backup Path Information and Finding the Shortest Backup Path are the two stages in the process. Finding the Shortest Backup Path

manages to determine the shortest routes between switches. In this step, the shortest path between any two witches is determined. The path between each switch (*Sx*) and its neighbor (*Sy*) through the middle switch (*Sm*) must be determined to obtain the shortest backup path. To accomplish this, we traverse through every switch and save every path on *Sm* that connects two switch neighbors. Algorithm 2 illustrates finding the shortest backup path after navigating all the switches. Here, switches are configured with the shortest backup path. This function is responsible for installing backup path information.

Algorithm 2: Backup Path Calculation Algorithm

Input: Network topology G = (S, E)
Output: Backup path set B for G

```
B ← empty set
sp ← floyd(G)
for each switch Sy ∈ S do
    for each neighbor switch Sx of Sy do
        if backup path for (Sy, Sx) does not exist in B then
            Sm← infinity
            path ← null
            for each switch s ∈ S where s ≠ Sy and s ≠ Sx do
                if len(sp(Sy, s)) + len(sp(s, Sx)) < Sm then
                    if (Sy, Sx) is not present in sp(Sy, s) and (Sy, Sx) is not present in
                    sp(s, Sx) then
                        Sm ← len(sp(Sy, s)) + len(sp(s, Sx))
                        path ← sp(Sy, s) + sp(s, Sx)
                    end if
                end if
            end for
            B ← B ∪ {path}
            B ← B ∪ {reverse(path)}
        end if
    end for
end for
return B
```

The algorithm for the network architecture G = (S, E) determines a backup path set. After initializing an empty set B, the Floyd algorithm determines the shortest path matrix. The process then runs through the network's switches, checking each neighboring switch for a backup path. The method then loops over all switches again to identify the shortest route between the current switch and its neighbor, removing the current switch and its neighbor from the path if a backup path is not included in set B. The path and its reverse are added to set B if it does not include the current switch and its neighbor. As a result, the algorithm returns the final set B.

4 Experimental Results and Discussions

This section delves into the pivotal phase of experimental results and discussions, comprehensively analyzing the outcomes of implementing and evaluating the proposed mechanism. This section presents and interprets the data gathered during experiments, shedding light on the proposed

mechanism's performance in diverse scenarios. This section is comprised of many subsections. The first section indicates evaluation metrics. The second section discusses the performance evaluation of proposed mechanisms. The third section describes the comparison of existing mechanisms.

4.1 Evaluation Metrics

The evaluation employs crucial metrics—recovery time, calculation time, and computational overhead—to measure the proposed mechanism's performance.

4.1.1 Recovery Time

The Recovery Time (RT) of link failure measures the time required by the switch to recover from the failure and continue the transmission of packets. The switch must obtain an alternative path to reroute packets if a link or a port fails. The switch of the failed link finds the backup path from its neighbors without notifying the controller.

4.1.2 Calculation Time

Calculation Time (CT) measures how long it takes to calculate every backup path between switches. The controller must install the appropriate backup path into each switch after calculating all switch paths to determine the shortest backup paths.

4.1.3 Computational Overhead

Computational Overhead (CO) in software-defined network link failure recovery encompasses resource utilization incurred by the software-defined network controller and network devices to handle link failures swiftly and efficiently. This metric is crucial for evaluating the efficiency of recovery mechanisms, emphasizing the need for optimized performance to ensure minimal disruption and maintain network integrity.

4.2 Performance Evaluation

The performance evaluation section focuses on assessing the efficiency of the proposed mechanism. This assessment is intricately divided into two categories: packet processing and path computation. These categories are designed to comprehensively evaluate the proposed mechanism, shedding light on its effectiveness in managing packet processing and path computation during link failure scenarios.

4.2.1 Packet Processing

A key component of the proposed mechanism is packet processing. It is employed to assess the efficiency with which the software-defined exchange point's multi-hop topology handles incoming packets before forwarding them to their designated destination.

The topology to assess the recovery time in the event of a connection failure is shown in Fig. 4. The switches are represented by circles with labels ranging from S1 to S8. Solid black lines that sequentially connect the switches represent the primary path. The backup path is A dotted red line connecting S1 and S2 directly. To enable P4, the study substituted bmv2 for Open vSwitch and utilized Mininet to build this architecture. Table 2 offers the topology information of Fig. 4. It outlines the connections between eight network switches, S1 through S8, categorizing them into primary and backup paths. Specifically, Switch S1 features a backup link to S2, designed to secure network continuity during

potential disruptions. The remaining switches, S3 to S8, are connected via primary paths, creating a robust topology that enhances data flow and network redundancy.

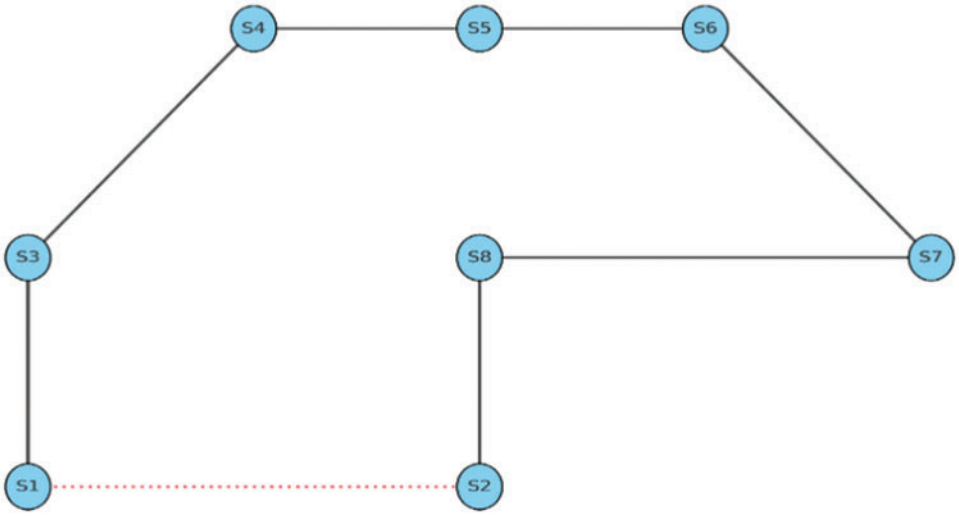

Figure 4: Network topology for evaluation link failure recovery time

Table 2: Topology information

Switch	Connected to	Path type
S1	S2	Backup
S2	S8	Primary
S3	S1, S4	Primary
S4	S3, S5	Primary
S5	S4, S6	Primary
S6	S5, S7	Primary
S7	S6, S8	Primary
S8	S2, S7	Primary

Table 3 compares the recovery times of the proposed mechanism and ENDEAVOUR mechanisms, showing that the proposed mechanism, with proactive backup path installation with P4 utilization, recovers from link failures more efficiently, with recovery times ranging from 19 to 51 ms. In contrast, ENDEAVOUR, which installs backup paths reactively, exhibits longer recovery times between 87 and 163 ms as the number of switches increases. The proposed mechanism utilizes P4 features for rapid packet forwarding upon failure, whereas ENDEAVOUR relies on OpenFlow features to handle affected packets after detecting a failure. The recovery times of ENDEAVOUR and the proposed mechanism for the number of switches in the backup path are displayed in Fig. 5. For the proposed mechanism, represented by a blue line, the recovery time begins at approximately 20 ms for 3 switches and rises moderately to just over 50 ms for 8 switches. This gradual incline on the chart suggests that ELFR's recovery time increases slowly as more switches are added, indicating a resilient performance against growing network complexity. In contrast, the ENDEAVOUR system, illustrated by a red line,

begins with a recovery time of just above 80 ms for 3 switches and more than doubles to around 160 ms for 8 switches. The ENDEAVOUR line reveals a sharper increase in recovery time with each added switch and potential variability in these times.

Table 3: Evaluation of packet processing

Number of switches in backup path	ELFR Recovery time (RT) (ms)	ENDEAVOUR Recovery time (RT) (ms)
3	19	87
4	24	96
5	30	113
6	34	129
7	41	144
8	51	163

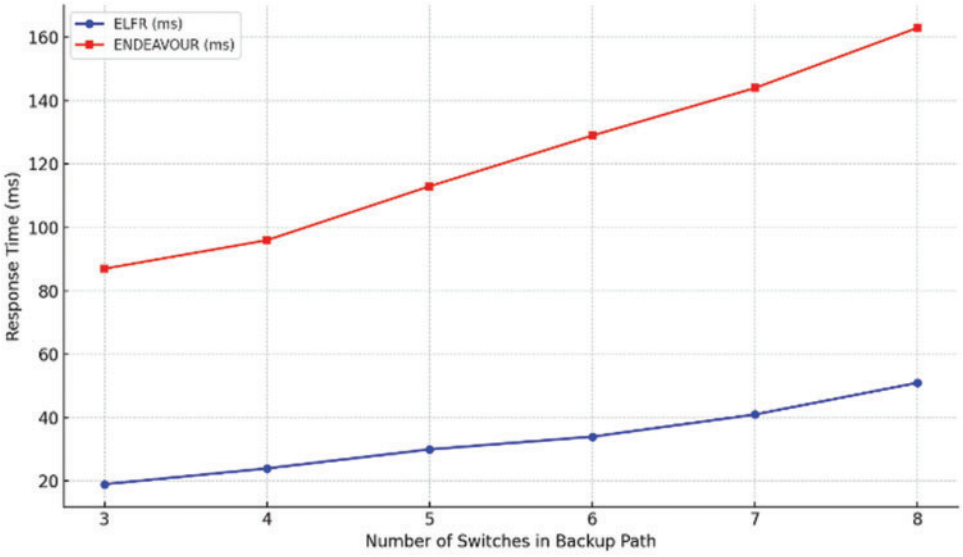

Figure 5: Evaluation of recovery time for packet

4.2.2 Path Computation

This assesses how long the controller will take to figure out every backup path between switches. To reduce the cost of switch storage and the computation time required by the software-defined network controller in the event of a link failure, this calculates a backup path for each neighbor switch of each switch. Each switch only keeps its neighbors' backup paths in case of a connection failure, and the controller installs the rules on the switches after calculations have been made. The controller will inform the appropriate switches about the backup path data via P4Runtime, facilitating communication between the switches and the controller.

Fig. 6 illustrates a network topology with 11 interconnected nodes centered around Node S1, which serves as a major hub connected to Nodes S2 and S3, enhancing network resilience and traffic

management. Nodes like S4 and S11 are peripheral endpoints which could be vulnerable to disruptions. In contrast, Nodes S5 and S6 are critical junctions with multiple connections that reinforce the network's routing infrastructure. Secondary pathways, such as S5 to S7, act as backup routes to ensure reliability. Table 4 outlines a network topology with Nodes S1 through S11, interconnected primarily in a circular formation from S1 to S10 and looping back to S3. This structure serves as the backbone for the main data traffic flow, optimising data distribution across multiple pathways to enhance the handling of large data volumes, thereby boosting reliability and efficiency. The network also integrates backup paths, specifically from S1 to S3 and S5 to S6, to maintain functionality during primary path failures or congestion. Notably, a disruption between S5 and S7 indicates the necessity of these backup routes for data rerouting in response to failures.

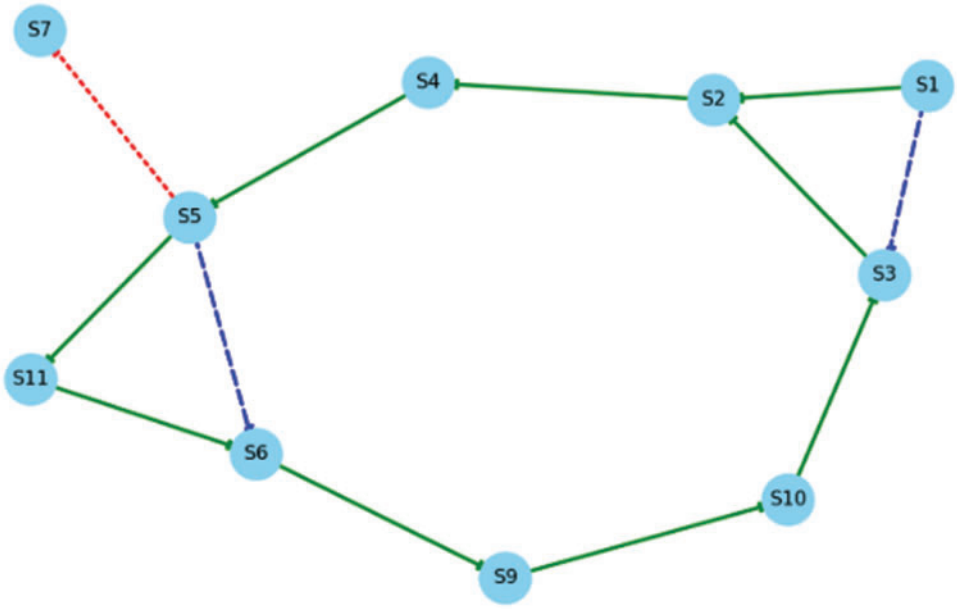

Figure 6: Topology evaluation for path computation

Table 4: Topology information

Nodes	Connected to	Path type
S1	S2	Primary
S2	S4	Primary
S4	S5	Primary
S5	S11	Primary
S11	S6	Primary
S6	S9	Primary
S9	S10	Primary
S10	S3	Primary
S3	S2	Primary
S1	S3	Backup

(Continued)

Table 4 (continued)

Nodes	Connected to	Path type
S5	S6	Backup
S5	S7	Failed

Installing routing table entries in S7, S10, and S9—a total of three entries—is the standard procedure for implementing the backup path in the topology. We can protect switch storage by using the well-proven proactive failure recovery technique in conjunction with the ELFR data plane. The ELFR data plane thus requires only one switch entry for every backup path. As shown in Fig. 6, we can save the entire backup path information (i.e., S7–S10–S9–S6) on Switch S7, and when connection S7–S5 breaks, we can add the path to the header of affected packets. Consequently, the following switches can transfer the packet without storing any information about the backup path by extracting the forwarding information from the packet header.

Table 5 compares the calculating times in milliseconds (ms) for the proposed mechanism and ENDEAVOUR mechanisms across networks with 2 to 7 nodes. The proposed mechanism consistently records faster computing times, increasing from 20 to 60 ms with the network size. In contrast, ENDEAVOUR starts at 45 ms for a 2-node network and climbs to 112 ms for a 6-node network but reduces to 66 ms for 7 nodes, suggesting improved efficiencies at larger scales. Overall, the proposed mechanism displays greater computational efficiency, particularly in smaller networks, making it a preferable choice for network management and optimization. Fig. 7 compares the calculating times in milliseconds (ms) for two mechanisms, the proposed mechanism and ENDEAVOUR, across networks varying from 2 to 7 nodes. The graph shows the proposed mechanism, depicted with a blue line, consistently maintaining lower computation times than ENDEAVOUR, represented by a red line. The proposed mechanism's computation time increases gradually, indicating stable scalability as the network size expands. Conversely, ENDEAVOUR's times start higher and rise sharply with network growth, except for a notable dip at 7 nodes, suggesting possible optimization or efficiency improvements at this specific network size.

Table 5: Evaluation of path computation

Number of nodes	ELFR Calculation time (CT) (ms)	ENDEAVOUR Calculation time (CT) (ms)
2	20	45
5	41	80
6	60	112
7	36	66

Additionally, the proposed mechanism quickly calculates backup paths by targeting nearby switches, reducing the number of paths to compute. Additionally, since backup routes connect neighboring nodes, finding alternatives is faster, enhancing the system's efficiency.

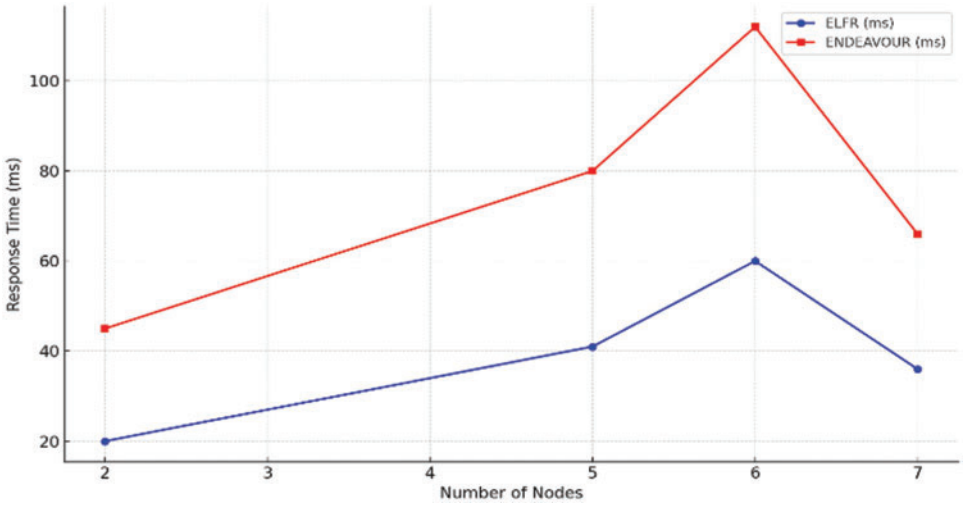

Figure 7: Evaluation of calculation time for path computation

4.2.3 Memory Storage

The ELFR mechanism incorporates an assessment of the memory storage overhead involved in path computation, aiming to minimize the additional memory required for processing routing information. This evaluation is significant for optimizing network performance, ensuring that routing is effective and resource-efficient, and conserving memory usage to facilitate a leaner network infrastructure.

Table 6 compares memory usage in megabytes (MB) for the proposed mechanism and the ENDEAVOUR mechanism across network Nodes S1 through S11. The proposed mechanism shows a minimal memory footprint, ranging from 0 to 2 MB per node, indicating its efficiency. In contrast, ENDEAVOUR consistently uses more memory, with usage between 2 to 4 MB per node. Notably, for Nodes S9 and S10, ELFR shows no memory usage, suggesting exceptional efficiency or non-engagement with these nodes, while ENDEAVOUR uses 2 MB at each. This data highlights the proposed mechanism as a more memory-efficient option, ideal for resource-sensitive networks. Conversely, ENDEAVOUR's higher consumption suggests a more feature-rich mechanism, offering enhanced capabilities at the expense of greater resource use. Fig. 8 graphically depicts the memory usage in megabytes (MB) of two routing mechanisms, the proposed mechanism and ENDEAVOUR, across network Nodes S1 to S11. The proposed mechanism consistently shows lower memory usage, generally maintaining a level range that implies a more efficient and effective routing approach. On the other hand, ENDEAVOUR's memory usage fluctuates significantly across the nodes and is overall higher, indicating the use of more complex algorithms or larger routing tables.

Table 6: Evaluation of computational overhead

Node	ELFR memory usage (Megabyte)	ENDEAVOUR memory usage (Megabyte)
S1	1	2
S2	2	4

(Continued)

Table 6 (continued)

Node	ELFR memory usage (Megabyte)	ENDEAVOUR memory usage (Megabyte)
S4	2	4
S5	1	4
S6	1	3
S7	1	2
S9	0	2
S10	0	2
S11	2	4

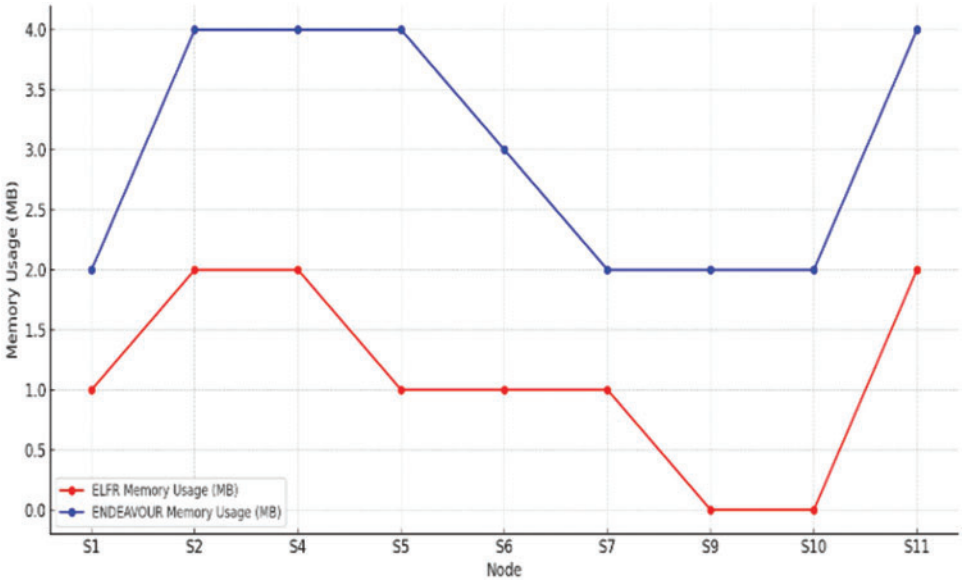

Figure 8: Evaluation of computational overhead

4.2.4 Comparison with Existing Mechanisms

Regarding recovery time for packet processing, computation time for path computation, and computational overhead for memory storage, the proposed mechanism is compared with current multi-hop software-defined exchange point-based link failure recovery mechanisms [20,21]. A comparison of the performance of two network techniques, the proposed mechanism and ENDEAVOUR, for recovery time (RT), calculating time (CT), and computational overhead (CO) is shown in Table 7. Enhanced Link Failure Rerouitng (ELFR) shows greater efficiency with lower metrics—37.5% for RT and 33.33% for CT and CO. Conversely, ENDEAVOUR requires more resources, shown by higher percentages of 62.5% for RT and 66.67% for CT and CO. Fig. 9 displays a bar chart comparing two network mechanisms, ELFR and ENDEAVOUR, across three metrics: Recovery Time (RT), Calculating Time (CT), and Computational Overhead (CO). Each mechanism is represented by three bars corresponding to these metrics. Enhanced Link Failure Rerouting (ELFR) consistently records lower percentages across all metrics, highlighting its greater efficiency than ENDEAVOUR.

Table 7: Comparison of proposed mechanism with existing mechanisms

Mechanism	Packet processing	Path computation	
	RT (%)	CT (%)	CO (%)
Enhanced link failure rerouting (ELFR)	37.5%	33.33%	33.33%
ENDEAVOUR	62.5%	66.67%	66.67%

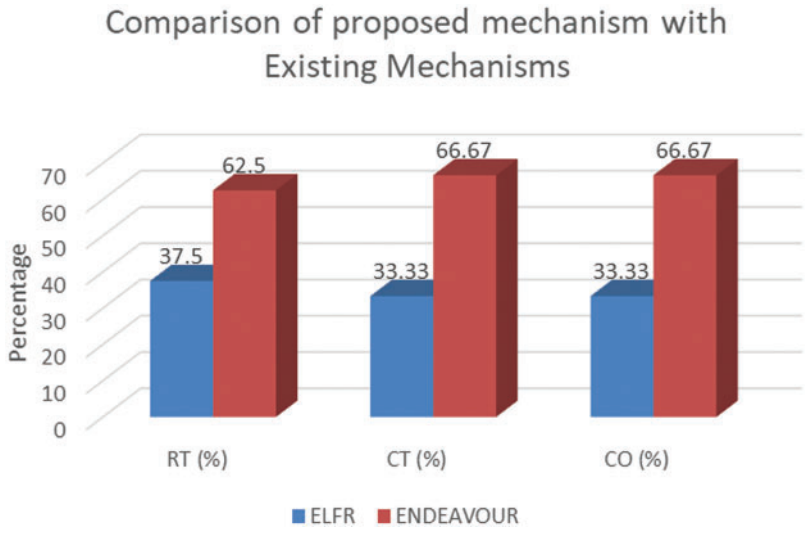

Figure 9: Comparison of ELFR with existing

5 Conclusion and Future Work

In conclusion, this paper proposed the Enhanced Link Failure Rerouting (ELFR) mechanism, which significantly improves managing multi-hop Software-Defined Exchange Points (SDXs). The proposed mechanism effectively reduces recovery time, enhances path computation efficiency, and optimizes switch memory usage by leveraging advanced algorithms and protocol-independent packet processors (P4). The evaluation results demonstrate that the proposed mechanism outperforms existing mechanisms, such as ENDEAVOUR, particularly in recovery time and computational overhead, making it a robust solution for complex software-defined exchange point environments.

Future work should explore extending the proposed mechanism's capabilities to handle multiple simultaneous link failures and address potential loop issues. Additionally, real-world testing under diverse network conditions is essential to validate the mechanism's robustness and adaptability. These efforts will help further refine the proposed mechanism and ensure its effectiveness in broader software-defined exchange point applications.

Acknowledgement: The authors thank SIMAD University, Somalia, for its financial assistance.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: Abdijalil Abdullahi contributed to the writing, design, implementation, and experimental result analysis of the proposed work. Selvakumar Manickam contributed to leading,

reviewing, and removing grammatical problems. Each author participated sufficiently in the production to take public responsibility for the relevant percentage of the content. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: This article does not involve data availability, and this section is not applicable.

Ethics Approval: Not applicable.

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

References

[1] A. Basit *et al.*, "Interconnecting networks with optimized service provisioning," *Telecommun Syst.*, vol. 73, no. 3, pp. 223–239, Aug. 2020. doi: 10.1007/s11235-019-00606-3.

[2] T. B. da Silva *et al.*, "Toward next-generation and service-defined networks: A NovaGenesis control agent for future Internet exchange point," *IEEE Netw*, vol. 36, no. 3, pp. 74–81, Jul. 2022. doi: 10.1109/MNET.008.2100555.

[3] V. Stocker, G. Knieps, and C. Dietzel, "The rise and evolution of clouds and private networks-internet interconnection, ecosystem fragmentation," in *TPRC49: 49th Res. Conf. Commun., Inform. Internet Policy*, Aug. 2021. doi: 10.2139/ssrn.3910108.

[4] P. Marcos, M. Chiesa, C. Dietzel, M. Canini, and M. Barcellos, "A survey on the current internet interconnection practices," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 50, no. 1, pp. 10–17, Mar. 2020. doi: 10.1145/3390251.339025.

[5] T. Hoeschele, C. Dietzel, D. Kopp, F. H. P. Fitzek, and M. Reisslein, "Importance of Internet exchange point (IXP) infrastructure for 5G: Estimating the impact of 5G use cases," *Telecomm. Policy*, vol. 45, no. 3, Apr. 2021, Art. no. 102091. doi: 10.1016/j.telpol.2020.102091.

[6] Y. Dabone, T. F. Ouedraogo, and P. J. Kouraogo, "Improving the linkage of Internet exchange points through connected ISPs ASes," in *Comput. Sci. On-Line Conf.*, Springer, Jul. 2022, pp. 180–187. doi: 10.1007/978-3-031-09073-8_17.

[7] T. Böttger *et al.*, "Shaping the Internet: 10 years of IXP growth," Oct. 2019, *arXiv:1810.10963*.

[8] L. M. Bertholdo *et al.*, "On the asymmetry of Internet eXchange points-why should IXPs and CDNs care?," in *2022 18th Int. Conf. Netw. Serv. Manag. (CNSM)*, Thessaloniki, Greece, IEEE, Dec. 2022, pp. 73–81. doi: 10.23919/CNSM55787.2022.9964817.

[9] D.Ó. Briain, D. Denieffe, D. Okello, and Y. Kavanagh, "Enabling models of Internet eXchange Points for developing contexts," *Dev. Eng.*, vol. 5, no. Sep. 2020, 2020, Art. no. 100057. doi: 10.1016/j.deveng.2020.100057.

[10] J. Chung, H. Owen, and R. Clark, "SDX architectures: A qualitative analysis," in *SoutheastCon 2016*, Norfolk, VA, USA, IEEE, Jul. 2016, pp. 1–8. doi: 10.1109/SECON.2016.7506749.

[11] M. Cevik *et al.*, "Towards production deployment of a SDX control framework," in *2022 Int. Conf. Comput. Commun. Netw. (ICCCN)*, Honolulu, HI, USA, IEEE, Sep. 2022, pp. 1–10. doi: 10.1109/ICCCN54977.2022.9868884.

[12] P. -W. Tsai, C. -W. Tsai, C. -W. Hsu, and C. -S. Yang, "Network monitoring in software-defined networking: A review," *IEEE Syst. J.*, vol. 12, no. 4, pp. 3958–3969, Dec. 2018. doi: 10.1109/JSYST.2018.2798060.

[13] J. Mambretti *et al.*, "Designing and deploying a bioinformatics software-defined network exchange (SDX): Architecture, services, capabilities, and foundation technologies," in *2017 20th Conf. Innov. Clouds, Internet Netw. (ICIN)*, Paris, France, IEEE, Apr. 2017, pp. 135–142. doi: 10.1109/ICIN.2017.7899403.

[14] A. Gupta *et al.*, "SDX: A software defined internet exchange," in *ACM SIGCOMM Computer Communication Review*, New York, NY, USA, ACM, Aug. 2014, pp. 551–562. doi: 10.1145/2740070.2626300.

[15] R. Birkner, A. Gupta, N. Feamster, and L. Vanbever, "SDX-based flexibility or internet correctness? Pick two!" *SOSR 2017-Proc. 2017 Symp. SDN Res.*, vol. 9, pp. 1–7, Apr. 2017. doi: 10.1145/3050220.3050221.

[16] A. Abdullahi, S. Manickam, S. Karuppayah, and M. A. Al-Shareeda, "Proposed enhanced link failure rerouting mechanism for software-defined exchange point," *Indones J. Electr. Eng. Comput. Sci.*, vol. 31, no. 1, pp. 259–270, Jul. 2023. doi: 10.11591/ijeecs.v31.i1.pp259-270.

[17] J. Ali, G. -M. Lee, B. -H. Roh, D. K. Ryu, and G. Park, "Software-defined networking approaches for link failure recovery: A survey," *Sustainability*, vol. 12, no. 10, May 2020, Art. no. 4255. doi: 10.3390/su12104255.

[18] Q. Li, Y. Liu, Z. Zhu, H. Li, and Y. Jiang, "BOND: Flexible failure recovery in software defined networks," *Comput. Netw.*, vol. 149, no. 5, pp. 1–12, Feb. 2019. doi: 10.1016/j.comnet.2018.11.020.

[19] A. Abdullahi, S. Manickam, and S. Karuppayah, "A review of scalability issues in software-defined exchange point (SDX) approaches: State-of-the-art," *IEEE Access*, vol. 9, pp. 74499–74509, Mar. 2021. doi: 10.1109/ACCESS.2021.3069808.

[20] M. Bruyere *et al.*, "Rethinking IXPs' architecture in the age of SDN," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 12, pp. 2667–2674, Dec. 2018. doi: 10.1109/JSAC.2018.2871294.

[21] G. Antichi *et al.*, "ENDEAVOUR: A scalable SDN architecture for real-world IXPs," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2553–2562, Nov. 2017. doi: 10.1109/JSAC.2017.2760398.

[22] J. Ali, G. Shan, N. Gul, and B. Roh, "An intelligent blockchain-based secure link failure recovery framework for software-defined internet-of-things," *J. Grid Comput*, vol. 21, no. 4, Oct. 2023, Art. no. 57. doi: 10.1007/s10723-023-09693-8.

[23] V. Muthumanikandan and C. Valliyammai, "Link failure recovery using shortest path fast rerouting technique in SDN," *Wirel. Pers. Commun.*, vol. 97, no. 2, pp. 2475–2495, Jun. 2017. doi: 10.1007/s11277-017-4618-0.

[24] T. Holterbach, S. Vissicchio, A. Dainotti, and L. Vanbever, "SWIFT predictive fast reroute," in *Proc. Conf. ACM Spec. Interes. Gr. Data Commun.*, Aug. 2017, pp. 460–473. doi: 10.1145/3098822.3098856.

[25] P. Mao, R. Birkner, T. Holterbach, and L. Vanbever, "Boosting the BGP convergence in SDXes with SWIFT," in *SIGCOMM Posters Demos 2017-Proc. 2017 SIGCOMM Posters Demos*, Aug. 2017, pp. 1–2. doi: 10.1145/3123878.3131965.

[26] M. Chiesa *et al.*, "PURR: A primitive for reconfigurable fast reroute," *ACM Conf. Emerg. Netw. Exp. Technol.*, pp. 1–14, Dec. 2019. doi: 10.1145/3359989.3365410.

[27] M. He *et al.*, "A rerouting framework against routing interruption for secure network management," *IEEE Access*, vol. 7, pp. 143620–143630, Oct. 2019. doi: 10.1109/ACCESS.2019.2945777.

[28] T. Semong *et al.*, "A review on software defined networking as a solution to link failures," *Sci. African.*, vol. 21, no. 18, Sep. 2023, Art. no. e01865. doi: 10.1016/j.sciaf.2023.e01865.

[29] S. Petale and J. Thangaraj, "Link failure recovery mechanism in software defined networks," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 7, pp. 1285–1292, Apr. 2020. doi: 10.1109/JSAC.2020.2986668.

[30] S. Kaur, K. Kumar, and N. Aggarwal, "A review on P4-Programmable data planes: Architecture, research efforts, and future directions," *Comput. Commun.*, vol. 170, no. 1, pp. 109–129, 2021. doi: 10.1016/j.comcom.2021.01.027.

[31] H. Miura, K. Hirata, and T. Tachibana, "P4-based design of fast failure recovery for software-defined networks," *Comput. Netw.*, vol. 216, no. 2, Oct. 2022, Art. no. 109274. doi: 10.1016/j.comnet.2022.109274.

[32] D. Wagner, M. Wichtlhuber, C. Dietzel, J. Blendin, and A. Feldmann, "P4IX: A concept for P4 programmable data planes at IXPs," in *Proc. ACM SIGCOMM Workshop Future Internet Rout. Address.*, Sep. 2022, pp. 72–78. doi: 10.1145/3527974.3545725.

[33] M. V. B. da Silva, A. S. Jacobs, R. J. Pfitscher, and L. Z. Granville, "IDEAFIX: Identifying elephant flows in P4-based IXP networks," in *2018 IEEE Global Commun. Conf. (GLOBECOM)*, Abu Dhabi, United Arab Emirates, IEEE, Feb. 2018, pp. 1–6. doi: 10.1109/GLOCOM.2018.8647685.

[34] J. Xu, S. Xie, and J. Zhao, "P4Neighbor: Efficient link failure recovery with programmable switches," *IEEE Trans. Netw. Serv. Manag.*, vol. 18, no. 1, pp. 388–401, Jan. 2021. doi: 10.1109/TNSM.2021.3050478.

[35] Z. Li, Y. Hu, J. Wu, and J. Lu, "P4Resilience: Scalable resilience for multi-failure recovery in SDN with programmable data plane," *Comput. Netw.*, vol. 208, May 2022, Art. no. 108896. doi: 10.1016/j.comnet.2022.108896.