Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2024.059050

REVIEW

A Survey of Link Failure Detection and Recovery in Software-Defined Networks

Suheib Alhiyari, Siti Hafizah AB Hamid* and Nur Nasuha Daud

Department of Software Engineering, Universiti Malaya, Kuala Lumpur, 50603, Malaysia

*Corresponding Author: Siti Hafizah AB Hamid. Email: sitihafizah@um.edu.my

Received: 27 September 2024 Accepted: 15 November 2024 Published: 03 January 2025

ABSTRACT

Software-defined networking (SDN) is an innovative paradigm that separates the control and data planes, introducing centralized network control. SDN is increasingly being adopted by Carrier Grade networks, offering enhanced network management capabilities than those of traditional networks. However, because SDN is designed to ensure high-level service availability, it faces additional challenges. One of the most critical challenges is ensuring efficient detection and recovery from link failures in the data plane. Such failures can significantly impact network performance and lead to service outages, making resiliency a key concern for the effective adoption of SDN. Since the recovery process is intrinsically dependent on timely failure detection, this research surveys and analyzes the current literature on both failure detection and recovery approaches in SDN. The survey provides a critical comparison of existing failure detection techniques, highlighting their advantages and disadvantages. Additionally, it examines the current failure recovery methods, categorized as either restoration-based or protection-based, and offers a comprehensive comparison of their strengths and limitations. Lastly, future research challenges and directions are discussed to address the shortcomings of existing failure recovery methods.

KEYWORDS

Software defined networking; failure detection; failure recovery; restoration; protection

1 Introduction

Traditional computer networks are inherently dynamic and complex, making their configuration and management a persistent challenge. These networks generally comprise numerous switches, routers, firewalls, and various types of middleboxes, all of which can experience multiple events simultaneously. Network operators are tasked with configuring the network to enforce various high-level policies and to respond to a diverse range of network events, such as link failures, traffic shifts, and security intrusions [1]. Therefore, Software Defined Networking (SDN) has been seen as a promising solution to address the challenges inherent in traditional computer networks [2]. SDN provides centralized control, global network visibility, and programmability, which are crucial for managing complex and large-scale networks [3]. Consequently, Carrier Grade Networks (CGNs) are turning to SDN technology in order to enhance their network management capabilities beyond those offered by traditional networks.

Despite the promise of enhanced network management offered by SDN technology, a range of new challenges have been introduced. One of the key challenges that SDN must address is ensuring efficient recovery from link failures [4]. Since a failure can significantly impact network performance and make services unavailable, reliability issues are crucial for the effective adoption of SDN. Failures in the network connections between forwarding devices in the data plane are the most frequent causes of link failures [5], with one link going down every 30 min [6]. Unexpected failures account for 80% of these occurrences, while only 20% are the result of scheduled maintenance [7]. A study conducted on Google's data centers and a set of wide area networks found that 80% of network component failures persisted for 10 to 100 min, leading to substantial packet loss [8]. Business sectors expect to lose more than $107,000 per hour due to data center down-time costs, and IT (Information Technology) outages result in a revenue loss of about $26.5 billion per year [9]. A network must return to service after a failure in a matter of a few tens of milliseconds to meet the requirements of a carrier grade network (i.e., a recovery time of under 50 ms) [10]. As an example, transport networks highly require reduced latency for voice and video conversations because voice echo occurs at a delay of 50 ms to avoid the inconvenience that these echoes cause during conversations [11].

In the SDN architecture, the control logic is moved from network devices into a centralized and separate control plane as depicted in Fig. 1 [12–14]. This architecture offers complete visibility of network entities, which in turn introduces new opportunities for efficient link failure recovery applications [15–17]. The enhanced visibility and centralization enable to improve network resiliency and ensure the effective implementation of SDN.

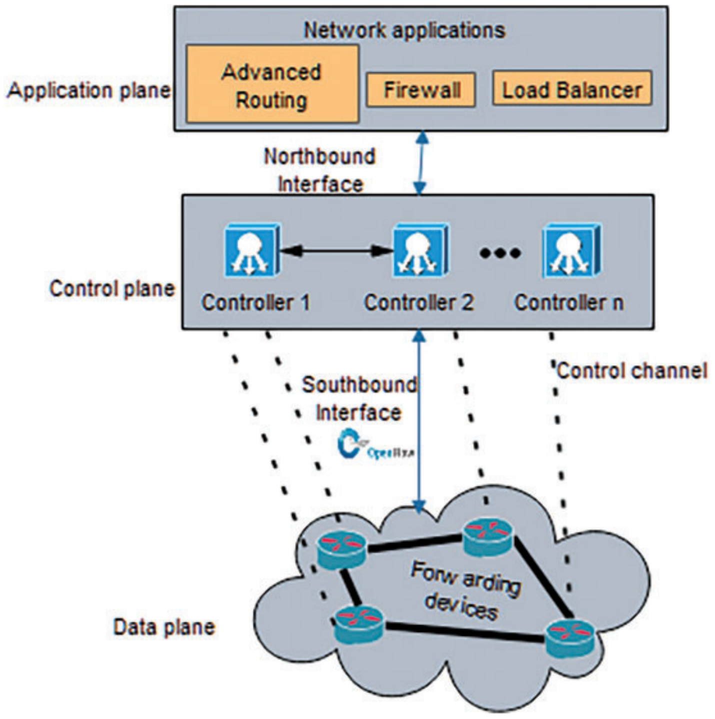

Figure 1: SDN architecture [18]

SDN-based link failure recovery mechanisms generally fall into two main categories: restoration and protection. In restoration, the controller plays an active role in the recovery process, whereas in protection, the data plane is capable of recovering from a failure independently, without the controller's intervention [5,19,20]. During restoration, the controller reactively addresses the link failure by calculating a new backup path and updating the necessary flow entries in the affected switches after receiving a failure notification from the data plane elements. The communication between the impacted switches and the controller, combined with the process of calculating a new route and reconfiguring the data plane, adds extra load on the controller and introduces delays in failure recovery. Consequently, the recovery time is heavily influenced by the scale of the network, often failing to meet CGN requirements for recovery time, which should be under 50 ms.

However, in the protection method, the flow entries of the backup paths are proactively installed at the data plane [21–23]. As a result, the performance of the protection approach is not affected by the number of disrupted flows. Additionally, it autonomously reroutes the affected flows without the controller intervention which enables the network to meet the recovery time requirement of CGNs. On the other hand, the protection mechanism consumes a much larger amount of the switches' Ternary Content Addressable Memory (TCAM), which is already limited, of the switches compared to the restoration. The majority of commercial OpenFlow switches, according to [24,25], have an on-chip TCAM with a size of between 750 and 2000 OpenFlow rules. But according to recent studies, modern data centers can have up to 10,000 network flows per second for each server rack [26]. Existing DCNs (Data Center Networks) provide a variety of critical and real-time services on a large scale [27]. A memory-aware and quick recovery of link failures between forwarding devices in the data plane, is a vital task and a fundamental requirement for increasing the reliability and robustness of these DCNs.

Although ensuring the reliability of SDN networks that host critical and sensitive services is a fundamental requirement for SDN adoption [28], the prevailing models for link failure recovery frequently struggle with one or both of these issues: extended recovery times for failures and considerable memory consumption in switches. On the one hand, restoration-based recovery adopts centralization of the recovery process at the controller. It uses switch memory efficiently; however, it requires a relatively long time to recover from failures which increases with large scaled networks because it requires communication with the controller to recover from the failure. Conversely, protection-based recovery decentralizes the recovery process and moves the recovery logic to the switches in the data plane. This approach ensures a small recovery time because it doesn't depend on the controller for the recovery process, but it significantly consumes the memory of the switches in the data plane because backup paths are installed proactively. Consequently, current recovery models demonstrate inefficiencies in performance and resource consumption under link failures. This paper provides insights into the ongoing research developments on fault management challenges in SDN networks, along with approaches aimed at detecting and ensuring the reliability of these networks.

The unique contributions of this survey are summarized as follows:

1. It provides a detailed comparison of existing failure detection techniques based on various criteria, including scalability, detection time dependency on network size, and compatibility with protection techniques.
2. It conducts a comprehensive analysis of restoration and protection-based recovery methods, highlighting their strengths and limitations, and compares them based on recovery time, scalability, processing overhead, and ability to meet CGN requirements.
3. It discusses key research challenges, such as scalability, latency, and resource constraints, and proposes future directions, including the integration of AI (Artificial Intelligence) and machine learning.

In this survey, we have carefully selected papers based on their relevance, recency, and contribution to the field of link failure detection and recovery in SDN. We focused on studies published between

2010 and 2024, ensuring that we included both foundational works and the latest advancements. The criteria for selecting suitable papers include:

- Relevance to link failure detection and recovery mechanisms in SDN.
- Publication in reputable journals and conferences, indicating quality and impact.
- Inclusion of both theoretical and practical contributions, covering various methodologies and technologies.
- Diversity in approaches, including both restoration and protection mechanisms, as well as hybrid methods.

The rest of the paper is organized as follows. In Section 2, we introduce the surveys that explored the failure detection and recovery in SDN and their contributions and limitations alongside highlighting how this survey differs from the existing surveys. In Section 3, we discuss the various link failure detection techniques and introduce a comprehensive comparison between them. In Section 4, the two most common approaches to recover from link failures, i.e., restoration and protection, are presented, and the restoration and protection-based studies are surveyed and compared. A comprehensive and critical comparison is conducted between restoration methodology and protection methodology to highlight each one's pros and cons. In Section 5, we first discuss the research challenges in the area of efficient failure recovery and then propose future directions to resolve these challenges, including emerging trends and technologies. Finally, in Section 6, we conclude the paper and highlight the main points of the survey, emphasizing the potential implications for network performance and real-world applications.

2 Existing Surveys on SDN Failure Recovery

Several surveys have explored fault management and failure recovery mechanisms in Software-Defined Networking (SDN), advancing our understanding while also revealing gaps that need further investigation. In this section, we critically review key surveys, discussing their contributions and limitations, and positioning our work within the field.

In [29], the authors conducted an early survey focused on fault tolerance in SDN, adapting detection and recovery techniques from traditional networking. They primarily examined data plane issues like link and switch failures, comparing restoration and protection methods. However, the survey's scope was limited, reflecting the early stages of SDN research. Moreover, the absence of the root cause analysis prevented deeper insights into failure mechanisms. Reference [30] offered a broader survey covering fault management across all SDN layers, from the data plane to the application plane. They categorized fault detection, localization, and recovery methods across these planes. Despite the extensive coverage, the lack of comparative analysis made it difficult to assess the effectiveness of various approaches.

Reference [31] provided a detailed survey structured around SDN's layered architecture, discussing fault tolerance across the data, control, and application planes. Their work highlighted contemporary challenges but missed several key aspects, such as incomplete coverage of hybrid techniques, and a lack of root cause analysis, limiting its potential to propose robust solutions. Reference [1] focused on link failure recovery, exploring restoration, protection, hybrid, and machine learning techniques. Although their survey provided detailed insights, it did not analyze the root causes of failures. These omissions limit the survey's applicability to more complex fault recovery scenarios. Reference [5] conducted a systematic review of data plane failure detection and recovery techniques in SDN. They classified current approaches, compared traditional networking with SDN, and discussed the root causes of failures. Additionally, the survey lacked a broader comparative analysis using newer performance metrics.

Despite the existing surveys on failure detection and recovery in SDN, there is still a need for a comprehensive and up-to-date analysis focusing specifically on the data plane mechanisms. Most existing surveys either focus on broader fault management aspects, security, or lack the inclusion of the latest research advancements up to 2024. Therefore, this survey aims to fill this gap by providing an in-depth investigation of node and link failure detection and recovery at the data plane, incorporating recent studies, and critically analyzing the shortcomings of existing approaches.

Unlike aforementioned surveys, this survey in deep investigate the node and link failure detection and recovery at the data plane. Fig. 2 shows the scope of this survey using the highlighted parts. Also, this survey critically highlights the shortcomings of existing detection and recovery approaches in order to enable researchers to address these shortcomings. In this survey, the approaches for detecting and recovering from link failures in Software-Defined Networking (SDN) are investigated. We compare existing failure detection techniques based on path failure detection, dependability of detection time on network size, scalability, ability to handle multiple failure locations, necessity for extra flow entries, capability to achieve link recovery time of less than 50 ms, compatibility with link protection techniques, and active or passive monitoring. The survey also demonstrates SDN-based link failure recovery using restoration and protection approaches. Additionally, we conduct a comprehensive comparison of restoration-based studies and protection-based approaches. Finally, a detailed comparison is conducted between protection-based approaches and restoration-based approaches in terms of recovery time delay, scalability, processing overheads at the controller and the switches, flexibility, congestion awareness, Ternary Content Addressable Memory (TCAM) consumption, and the ability to meet Carrier Grade Networks (CGN) requirements.

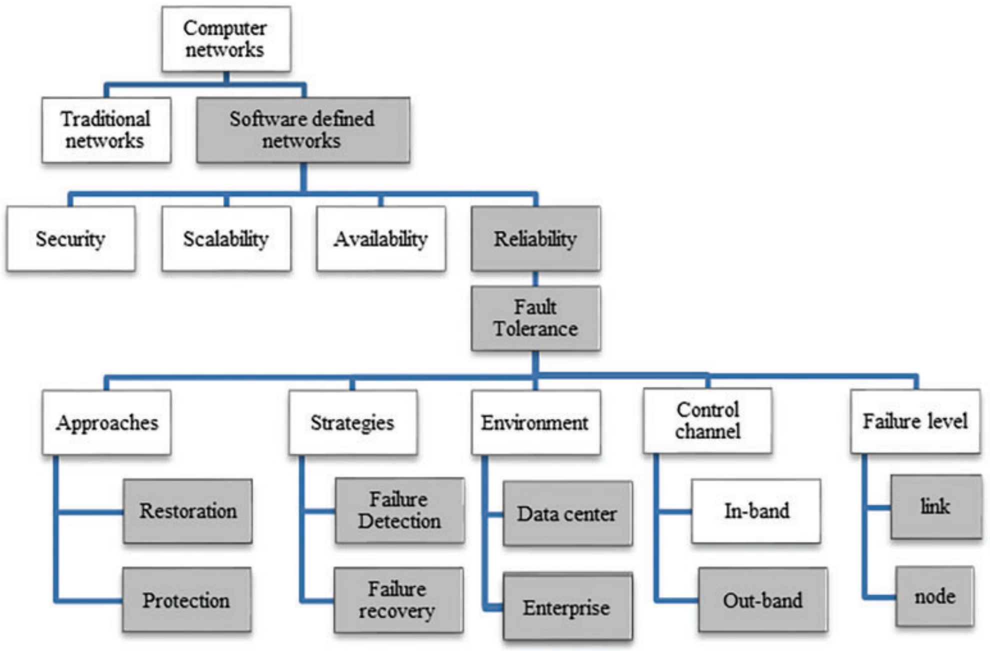

Figure 2: Scope of the survey

3 Link Failure Detection Techniques in SDN

The first step to recovery from failures is to detect that there is a failure. Thus, the speed at which a failure is detected will directly influence the overall recovery speed. Several kinds of failures can affect a network state. Nodes can crash, links can break, either as a result of a physical issue (e.g., submarine cable disruption) or of a logical issue (e.g., bringing an interface down on a router's administrative interface), resulting in flows that are no longer behaving properly and are losing packets. Fig. 3 depicts the classification of detection mechanisms that have been used in SDN networks.

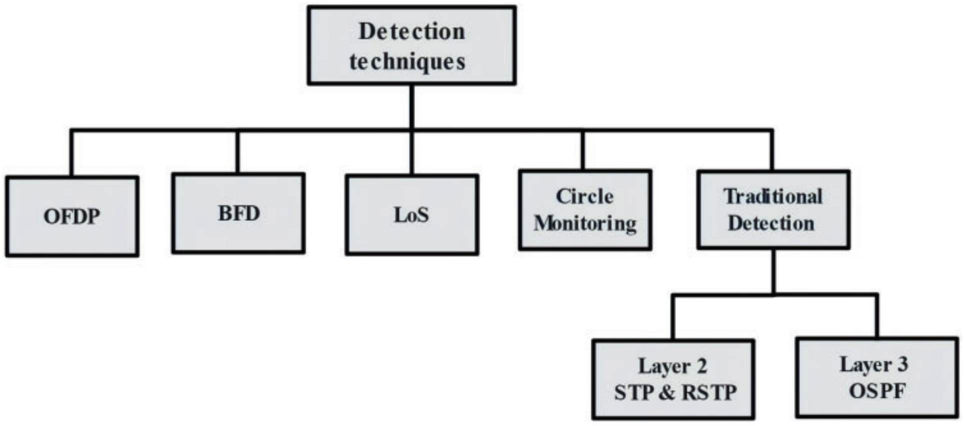

Figure 3: Detection techniques in SDN

3.1 OpenFlow Discovery Protocol (OFDP)

Although there is no standard protocol in SDN to discover the links between switches in the data plane, most SDN controllers use OFDP protocol for link discovery [32–34]. SDN controllers such as OpenDaylight [35], Floodlight [36], POX [37], Ryu [38], Beacon [39], Cisco Open SDN Controller [40] and Open Network Operating System (ONOS) [41,42] utilize OFDP for link discovery.

To discover links between switches, the OFDP protocol uses a modified version of the LLDP (Link Layer Discovery Protocol) frame format [43]. In traditional networks, the LLDP protocol is commonly used to discover links between switches. However, unlike OFDP, LLDP does not rely on centralized control, with switches sending and receiving LLDP advertisements independently. Despite this difference, OFDP uses the structure of LLDP frames with a few changes, but it operates differently in order to conform to the SDN architecture, which centralizes control logic at the controller. As a result, in OFDP, switches do not initiate LLDP advertisements; rather, the controller manages the entire link discovery process.

Fig. 4 shows that when Switch 1 receives a Packet_Out OpenFlow messages, it decapsulates the LLDP packet from the Packet_Out message, and forwards only the LLDP packet through each corresponding port. When Switch 2 receives an LLDP packet, it parses the packet and records the chassis ID (Identifier) and ingress port ID. Switch 2 then encapsulates the LLDP packet in a Packet_In message and sends it to the controller, using a pre-installed flow entry from its flow table. The controller then parses the Packet_In message to identify the discovered link, which is expressed by the mapping between (Switch 1 chassis ID, port ID) and (Switch 2 chassis ID, port ID) [44]. This process identifies the link in one direction. To discover the reverse direction of the link, the same procedure must be followed [32].

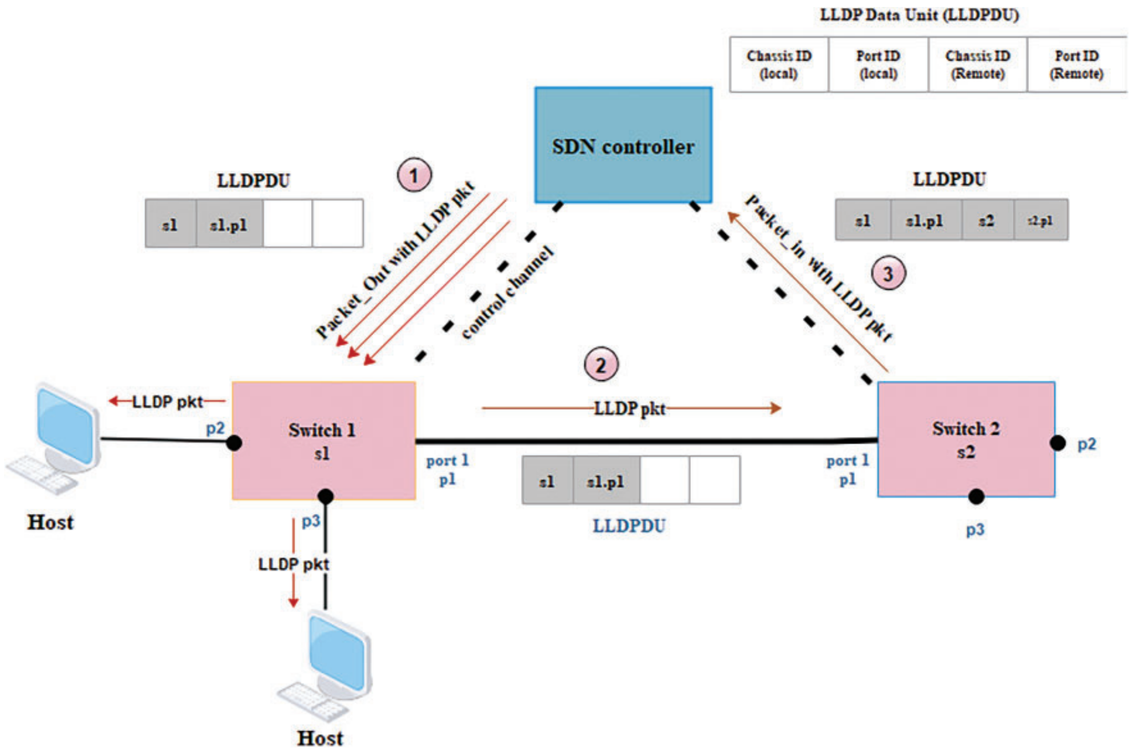

Figure 4: OFDP protocol methodology [32]

As previously stated, the controller uses OFDP for topology discovery and will send an LLDP packet that is encapsulated in a Packet_Out message to all active ports in the network at regular intervals (10 s). Such a discovery mechanism could have serious implications for the performance and functionality of SDN networks, particularly large networks. These issues can be concluded as follows [45]:

(1) Overhead on the controller and control channel

For large networks with many switches, the controller sends a large number of Packet_Out messages during each discovery interval, which can put a significant overhead on the controller's CPU (Central Processing Unit) [32,42,46] and the control channel, particularly when in-band control channels are used [32].

(2) Inefficient link failure detection

There are no mechanisms in the OpenFlow protocol to notify the controller of network failures. The OpenFlow protocol implies a mechanism in the form of an OFPT_PORT_STATUS message, which is sent to the controller when an OpenFlow switch port is administratively turned up or down [34,47]. Thus, most controllers will learn about link failures in the next discovery round, which takes about 10 s [34,42]) [48]. This is deemed too long for dynamic environments where topology changes occur frequently at short intervals [1,34]. Such a delay significantly impacts the performance of applications that depend on the topology information discovered by the controller for their operations [49].

3.2 Loss of Signal (LoS)

LoS is widely used in carrier-grade networks to detect link failures [50]. When the switch detects Loss of signal for one of its ports, it will inform the controller using OFPT_PORT_STATUS message with “down” state [51]. Likewise, when it detects a signal from one of its ports, it will send OFPT_PORT_STATUS message with “UP” state. LoS can be used to detect local link failures but not path failures. Because of that it may be used as a detection method for “per-link” based protection [22].

LoS relies on the IEEE 802.3 standard, which outlines a PHY-level detection mechanism for Ethernet PHY (Physical Layer). According to this standard, the transmitting device periodically sends a simple heartbeat pulse, known as a Normal Link Pulse (NLP), typically every 16 $\pm$ 8 ms. If the receiving device does not detect either a data packet or an NLP within a specified time window (usually 50–150 ms), the link is considered failed by the receiving device. Due to its slow detection rate, LoS is unable to meet the delay requirements of Carrier Grade Networks (CGNs), which demand a response time of less than 50 ms.

3.3 Bidirectional Forwarding Protocol (BFD)

BFD is a low-overhead protocol that is designed to monitor the liveness of links or paths on contrary to LoS which is used only for local links liveness monitoring [52]. Furthermore, BFD detects the link failures with adaptable and very low latencies for any media, at any protocol layer. Also, BFD detects failures in any type of connection between systems such as direct physical links, virtual circuits, tunnels, and unidirectional links.

BFD operates in two modes: asynchronous and demand. In asynchronous mode, which is the primary mode, the BFD generator on each system periodically sends control messages to the opposite end of the BFD session. If a specified number of control messages are missed, the BFD session is marked as “down.” In demand mode, it is assumed that the system has another independent method of confirming connectivity with the other system. Once a BFD session is established, the system may request that the other system to stop sending BFD control packets, except when it needs to explicitly verify connectivity. In such cases, a brief exchange of BFD control packets takes place, after which the far system remains quiet.

The detection time of link events in BFD depends on the transmission interval of the control packet (T_i) and the detection multiplier (M). T_i is the frequency of the control messages and M is the number of control packets that should be lost before considering the neighbor end-point unreachable. Failure detection time is given by Eq. (1):

$$T_{det} = M * T_i, \tag{1}$$

The transmit interval T_i is lower-bounded by the Round-Trip Time (RTT) of the link [52]. In order to achieve a detection time of 50 ms, the values of $T_i = 16.7$ ms and a detection multiplier of $M = 3$ are enough.

3.4 Circle Monitoring

In [53], the authors propose a path preplanning scheme that takes advantage of interface specific forwarding (ISF) to resolve the problem. They cover all of the links of the networks with at least one monitoring path. Every specific time, the probes during Path Alive Monitoring (PAM) traverse through each monitoring path. These probes start and end to the controller. If the monitoring probe does not return to the controller through a predefined interval, then there is a failure occurred. The Failure Location Identification (FLI) will start in this round, an FLI probe traverses through the path, and the receiving node will send the FLI probe to the next hop and send at the same time a copy of the probe to the controller. if there are the copy is not received by the controller, one of the adjacent of the failed link is identified. The result of 50 ms monitoring period showed 100 ms for recovery in the worst case.

To reduce the packet loss probability and the jitter, bandwidth is reserved for the monitoring round to avoid drops of packets due to congestion or other conditions and then generating false alarms. They use the QoS querying mechanism that is implemented in different OpenFlow switches to guarantee the bandwidth. One of the most important objectives in the approach is to find the minimum number of monitoring cycles that cover all links with minimum number of hops to minimize the delay and jitter. Therefore, they find the path that covers all of the links based on the graph theory, and divide this path to multiple sub-paths, with each path does not exceed more than h hops.

In [54], the proposed algorithm finds the minimum number of cycles that cover all the links in the network. It uses three types of probes to detect, locate and find the type of failure (Node/link). After detecting the failure by monitoring probes, two types of probes are sent through the cycle: failure location probes to find the failure location, and the second is type detection probes to know the type of failure: link or node. More than one controller is utilized: one of them is master in active mode, and the others serve as secondary controllers in standby mode.

In [55], it proposes a new approach to notify the controller of any failures that occur outside its designated region, in case multiple controllers are employed to manage large-scale networks and packets can traverse arbitrary links. Similarly, new approach is proposed by [53]. However, a key distinction is that upon detecting a fault, a binary search is utilized, progressively narrowing the search area until the specific broken link is identified. Despite this, both approaches still require additional flow table entries either to return packets to the controller or to reroute them.

3.5 Traditional Detection Mechanisms

For TCP/IP (Transmission Control Protocol/Internet Protocol) networks, the detection mechanism can be defined based on the layer that is operating. For Datalink layer, some of the popular protocols for link failure detection in Datalink layer are STP (Spanning Tree Protocol) and RSTP (Rapid Spanning Tree Protocol) [56]. All of these protocols are classified as slow; the detection period is in seconds and does not meet the delay requirements for most of the applications. While in Network Layer, OSPF (Open Shortest Path First) is considered one of the most popular and efficient routing protocol [57]. OSPF maintains the status of the shortest paths between forwarding devices in the network and updates them in case of the occurrence of link or device failures. It uses hello packets every relatively large interval which leads to exceeding the order of milliseconds and therefore OSPF is not suitable for CGNs [52,58].

3.6 Comparison between Detection Mechanisms

When comparing different detection mechanisms, it can be found that only BFD and circle monitoring can detect failures per path while the other only detect failures per links only. The detection time in OFDP, circle monitoring and traditional detection, increases when the network size increases. In LoS and BFD, the detection time does not depend on the network size, which makes LoS and BFD more scalable than other mechanisms. Regarding the capability of detecting multiple failures, all of the approaches can detect multiple failures except BFD and circle monitoring. Also, only LoS and

BFD do not require extra flow entries to be installed in the forwarding devices in the data plane to work properly. In addition, only BFD can be used as detection mechanism to achieve less than 50 ms of overall recovery time which is not possible with other mechanisms. Also, it is found that LoS and BFD are the only detection methods that have been used in the protection mechanisms to monitor the watch ports in the group buckets. Finally, LoS is the only passive detection mechanism meaning that it does not send any probes to check the status of links. Table 1 introduces a detailed comparison between the different link failure detection mechanisms.

Table 1: Comparison between different detection mechanisms

	OFDP	LoS	BFD	Circle monitoring	Traditional detection
Path failure detection	No	No	Yes	Yes	No
Dependability of the detection time on the network size	Dependent	Independent	Independent	Dependent	Dependent
Scalability	Not scalable	Scalable	Scalable	Not scalable	Not scalable
Multiple failure location	Yes	Yes	No	No	Yes
Need extra flow entries	Yes	No	No	Yes	Yes
Ability to achieve less than 50 ms of link recovery time	Not able	Not able	Able	Not able	Not able
Have been used with link protection techniques?	No	Yes	Yes	No	No
Active or passive monitoring	Active	Passive	Active	Active	Active

4 Link Failure Recovery Mechanisms in SDN

Once a failure has been detected, the network has to react and recover from this failure and ensure the connectivity between all hosts. There are two types of reactions that are possible, depending on the timescale at which the recovery process occurs. Generally, the controller can do link failure recovery in two general approaches: protection and restoration. In protection, backup routes are configured in advance of a failure. It calculates the backup paths and configures the flow tables of the corresponding switches with the required flow entries. Conversely, in restoration, backup paths are computed only after the failures occur by calculating a new path to restore the disrupted flows and install the required flow entries. In the following subsections, we will discuss the two approaches in detail, introduce the studies that adopt them and present a comprehensive comparison between the approaches.

4.1 Restoration Methodology

The process of restoration passes into three steps: first, the affected switches of the link failure inform the controller about the change in a port's status. Then, the controller calculates an alternate path to recover the affected flows that use the failed link. At last, the controller sequentially installs new flow entries, modifies or removes flow entries from the corresponding OpenFlow switches.

Fig. 5 presents an example of the restoration methodology. When the link between SW2 and SW3 fails (1), SW2 and SW3 will inform the controller about the failed link (2). After that, the controller will compute another path to reroute the traffic between SW2 and SW3 and update the flow tables of them (3). The network will use the new path and recover from failure (4).

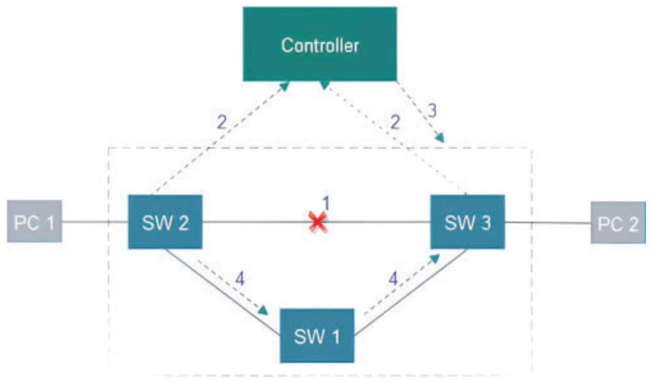

Figure 5: The methodology of restoration approach

Restoration-based studies

In this subsection, the approaches that adopt the restoration are introduced and discussed focusing on their aims, detection methods, performance metrics, and limitations. Table 2 introduces a comprehensive comparison between restoration-based approaches.

Reference [19] proposed a restoration approach using Loss of Signal (LoS) as detection technique to recover from link failures in OpenFlow networks. Upon receiving a port down notification, the controller recomputes new paths for affected flows. While the proposed approach is effective in small-scale networks, there are still key concerns about scalability and recovery time in larger networks. Similarly, Reference [59] proposed a recovery approach based on cycle structure. This approach first computes a tree for the topology. For each link that is not included in the tree, the approach assigns a tie-set (i.e., actually a circle) for such links. Hence, if a link fails, the algorithm can find and use the assigned tie-set to recover from the failure.

In contrast, Reference [60] introduced a scalable failover method for large-scale data center networks. The proposed method focused on choosing specific switches from different network layers in order update their flow tables to recover from the failures. The results show reducing the recovery time and improving the scalability.

Furthermore, in [61], the researchers proposed a method to decrease the restoration time of multi-link failures by minimizing the cost of flow operations and reducing the number of flow table updates. This method employs a Dijkstra-like algorithm to identify the shortest path with minimal operational costs, focusing on metrics such as operation cost and path cost. However, the study did not conduct an evaluation of the recovery time, leaving this aspect unexplored. Similarly, Reference [20] proposed a restoration-based approach that uses the shortest path first algorithm. The approach ensures the

QoS for high-priority traffic by ensuring minimum delay. However, the approach is not feasible for large-scale SDNs due to the complexity of the algorithm used to discriminate traffic significance.

Reference [62] proposed a method that depends on minimizing the number of flow entries that need modification during restoration. The proposed method precomputes multiple shortest paths and chooses the routes that require minimal flow updates, which reduces the operational costs associated with flow entry modifications. However, the use of longer alternate paths may increase communication delays.

In a related manner, Reference [63] proposed the Local Fast Reroute (LFR) technique, which utilizes the VLAN (Virtual Local Area Network) tagging method to aggregate multiple flows into a single large flow. While LFR reduces the number of flow entries and updates required for recovery, it may decrease the granularity of flow control. In another approach, Reference [64] aimed to reduce the computational overhead of path recalculation on the controller by adopting pruned searching algorithms from graph theory. This enhances scalability and reduces recovery time in large-scale networks, despite the impact on operational costs of flow modifications which requires further evaluation.

In [65], the proposed approach addressed the challenge of recovering from nested failures in scenarios where group-based protection by VLAN ID is used. They proposed the use of Bidirectional Forwarding Detection (BFD) and adopted pruned searching from graph theory to calculate the shortest path for recovery. The evaluation focused on three key metrics: recovery time, the number of flow entry updates, and the number of flows successfully restored. Although the proposed method demonstrated effectiveness, the evaluation was conducted using a small testbed, which may limit the generalizability of the results.

Finally, Reference [24] enhanced failure recovery in SDNs by introducing a system that leverages flow table awareness and advanced flow classification. It employs modules like Topology Discovery, Flows Handler, Flow Table Capacity, and Path Computation and Recovery to ensure robust network resilience. The proposed solution significantly reduces packet losses to less than 3%, decreases round trip times, and increases throughput, outperforming traditional methods.

Table 2: Comparison between restoration-based approaches

Ref.	Aim	Detection	Method	Performance metrics	Limitation
[19]	To recover from link failures in openflow networks	LoS	When receiving a port_down notification, the controller recalculates paths for affected flows	Recovery time	The testbed and workload are too small
[66]	Develop a scalable, efficient fault-tolerant system for SDN that quickly recovers from data plane failures	LoS	CORONET uses topology discovery for real-time status and calculates multiple link-disjoint shortest paths	Recovery time	The recovery time does not meet the requirements of 50 ms in CGNs

(Continued)

Table 2 (continued)

Ref.	Aim	Detection	Method	Performance metrics	Limitation
[59]	To implement a quick failure recovery approach that minimizes packet loss in large, complex networks	N/A	Computes a topology tree and assigns tie-sets to non-tree links for failure recovery	Recovery time	Increase using of flow tables of all switches
[60]	Recover from link failures in a locally efficient way	LoS	Relocate the impacted flows based on connectivity matrix tables and traffic data	Recovery time	The detection time is too long
[61]	Reduce multi-link failure restoration time by minimizing flow operations and flow table updates	LoS	The method introduces a Dijkstra-like algorithm to find the shortest path with minimal costs	1. Operation cost 2. Path cost	No evaluation conducted for recovery time
[20]	To calculate paths that introduce minimum delay for high priority packets	LoS	Shortest Path First Calculation and QoS for high priority traffic	Recovery time	The algorithm that is used for path calculation is not scalable
[63]	To recover quickly with a smaller number of flow entries and flow updates	LoS	The proposed Local Fast Reroute (LFR) to quick recover from link failures by aggregating the flows in a one big flow by Vlan tagging	1. Recovery time 2. No. of updated flow entries 3. No. of total flow entries	Aggregation of flows reduce the granularity of flow control
[62]	To reduce the number of flow updates	LoS	A graph theory-based reactive recovery for single link failures using the longest shortest path to minimize costs	1. Recovery time 2. Number of flow entries installed 3. Operation cost	High packet loss because the increased length of the calculated path

(Continued)

Table 2 (continued)

Ref.	Aim	Detection	Method	Performance metrics	Limitation
[54]	To detect link/node failures in for out-band or in-band control channels networks and recover the network from failures within less than 50 ms	Circle monitoring	The proposed algorithm finds the least number of cycles that covers all the links in the network	Recovery time	1. The experiments do not reflect the performance in terms of the recovery time for large networks 2. The testbed used is too small
[64]	To increase the recovery time by accelerating the recovery path calculations	LoS	Adopting pruned searching from the graph theory to calculate the shortest path	Recovery time	Operation cost not evaluated
[65]	To recover from nested failure when group-based protection by vlan ID is used	BFD	Adopting pruned searching from the graph theory to calculate the shortest path	1. Recovery time 2. Number of flow entries updates 3. Number of flows that are successfully restored	Small testbed used for evaluation
[67]	To develop SafeGuard to enhance SD-WAN (software-defined networking in a wide area network) recovery by optimizing bandwidth and switch memory to reduce congestion and boost efficiency	LoS	Use OpenFlow Fast Failover Groups to instantly detect link failures and reroute traffic via pre-installed backups, without controller intervention	1. Number of congested links 2. Length of backup routes 3. Overall link utilization in the network	1. Potential challenges in scaling the heuristic for very large networks due to its NP (Non-deterministic Polynomial)-hard nature. 2. The need for empirical tuning of the optimization model's parameters to balance multiple objectives

(Continued)

Table 2 (continued)

Ref.	Aim	Detection	Method	Performance metrics	Limitation
[24]	To reduce the size of TCAM memory used for link failure protection and the link recovery time	LoS	Emphasize flow table awareness and classification by integrating topology discovery, flow management, capacity monitoring, and path computation	1. Percentage of Packet Losses 2.Round Trip Time (RTT) 3. Throughput	Needs too much processing at the controller

4.2 Protection Approach

In the protection approach, the backup paths alongside with working paths are preconfigured before any occurrences of network failures. Hence, if a link failed, the disrupted flows are recovered using backup paths immediately without involving the controller. Fig. 6 explains the methodology of protection. When the link between SW2 and SW3 fails (Step 1), SW2 without involving the controller will switch to the pre-installed backup path (BP) (Step 2). Thus, the network will use the backup path and recover from failure (Step 3).

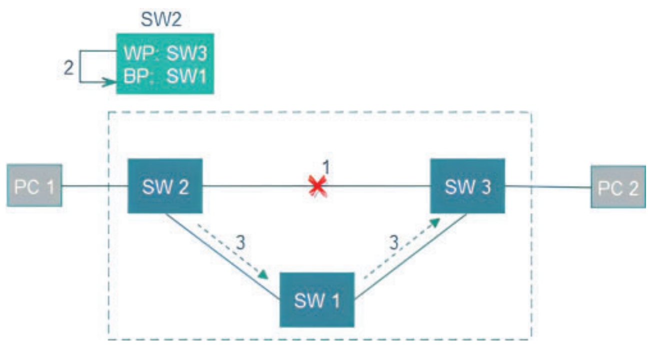

Figure 6: The methodology of protection approach

4.2.1 Level of Protection

The protection approaches can be divided based on the portion of the network that is protected into the following three levels of protection [17]:

(A) Path-based recovery

In this type, the controller will use the updated topology information and compute a disjoint backup path between each two hosts. The problem is that no full backup path will be configured at the data plane, when it is impossible to find a disjoint path.

(B) Link-based recovery

In this level of protection, a sub-path for backup is computed for each link alongside the working path. The sub-path is defined as a path between the switches located at the two ends of the link.

(C) Segment-based recovery

A backup sub-path is computed for each segment of the working path. A segment is defined as a series of consecutive links alongside the working path. It begins and ends at nodes that have backup paths that are fully disjointed from the remainder of the segment. Fig. 7 depicts the protection approaches based on the level of protection.

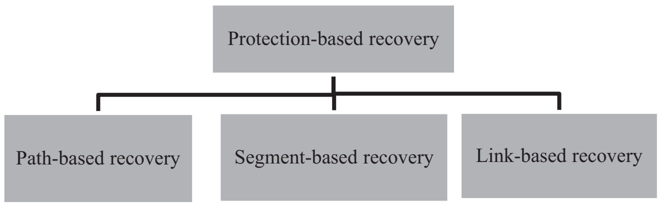

Figure 7: Classification of protection approaches based on the protected portion

Different levels of protection: per-path, per-segment, and per-link, perform differently based on the following metrics [17]:

(1) Failure recovery time

Path-based recovery necessitates communication of the failure between the controller and the source nodes of the paths in order to switch over to backup paths. In contrast, link-based recovery eliminates the need for signaling upon failure, because nodes can autonomously detect the failure and then recover from it. This makes it the fastest approach to recovering from failures. Meanwhile, segment-based recovery reduces signaling requirements but it still needs to notify the first node in the segment to reroute the flows to the backup path.

(2) Dependency on cranckback routing

In the context of path protection, the initial node where the working path connects with the backup path is the source node. However, in link- and segment-based protection, this node may be located closer to the destination. Crankback routing causes the links to be used twice during a failure, which can lead to congestion and packet loss. In contrast, link-based protection can reduce the need for crankback routing during T_F, thereby preventing congestion and excessive link utilization during the failure period.

(3) Number of paths

For 1:1 path-based protection, each flow requires two paths to be computed. One limitation of link-based protection is the need to compute many paths. Paths must be computed for each link alongside the working path between the source and destination when using 1:1 link-based protection. The total number of path computations required is $2 + (P - 2) = P$, where P is the working path length. Table 3 presents comparison between different protection levels.

4.2.2 Protection-Based Studies

Overall, the aim of the protection-based approaches is to achieve fast link failure recovery in SDN networks by proactively installing backup paths prior to link failure. This survey classifies the approaches according to the methodology used, as it is depicted in Fig. 8. A comprehensive comparison between protection-based approaches in Table 4.

Table 3: Comparison between different Protection levels

	Recovery time	Dependency on cranckback routing	Number of paths
Path-based recovery	Large	Highly dependent	Little
Segment-based recovery	Moderate	Moderately dependent	Moderate
Link-based recovery	Little	Independent	High

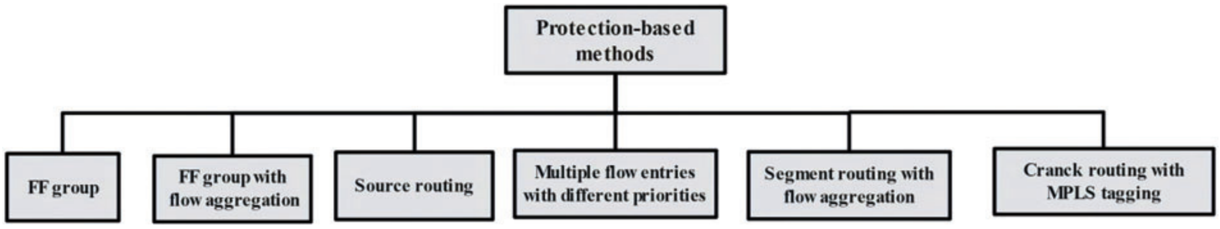
Protection-based methods
FF group
FF group with flow aggregation
Source routing
Multiple flow entries with different priorities
Segment routing with flow aggregation
Cranck routing with MPLS tagging

Figure 8: Classification of the protection approaches based on the methodology

Fast Failover group

Beginning with OpenFlow version 1.1, group table functionality was introduced to overcome the limitations of the functionality of flow entries, allowing to execution of various operations on the packets that were previously impossible with flow entries [68]. There are four types of group entries in OpenFlow 1.1 and afterward: ALL, SELECT, INDIRECT, and FAST FAILOVER (FF) [51]. The FF type is particularly designed to monitor the status of links and facilitate failover to a backup path. Fig. 9 shows the components of Fast Failover groups.

Fig. 9 depicts how the group table is configured with multiple action buckets, each linked to a switch port [51]. Every action bucket is assigned a priority, which distinguishes between working and backup paths. If no priority is assigned, the buckets' input order determines the priority. The status of the link or path associated with each output port is continuously monitored, either passively using OpenFlow's Loss-of-Signal (LoS) detection or actively via Bidirectional Forwarding Detection (BFD). If a link is down, the associated action bucket is considered down, causing the Fast Failover group table to switch to the next active action bucket, which contains the actions to recover the disrupted flows. OpenFlow switches configured with FF group tables that include protection paths can perform failover autonomously without requiring interaction with the controller.

References [69,70] and [22] leveraged the Fast Failover (FF) group tables in OpenFlow switches for autonomous and fast recovery alongside BFD for rapid link failure detection. By proactively installing backup paths in switches, the proposed approach achieves recovery times within the CGN requirement of less than 50 ms. However, the increased TCAM usage due to additional flow entries is a concern for limited-size switch memory resources.

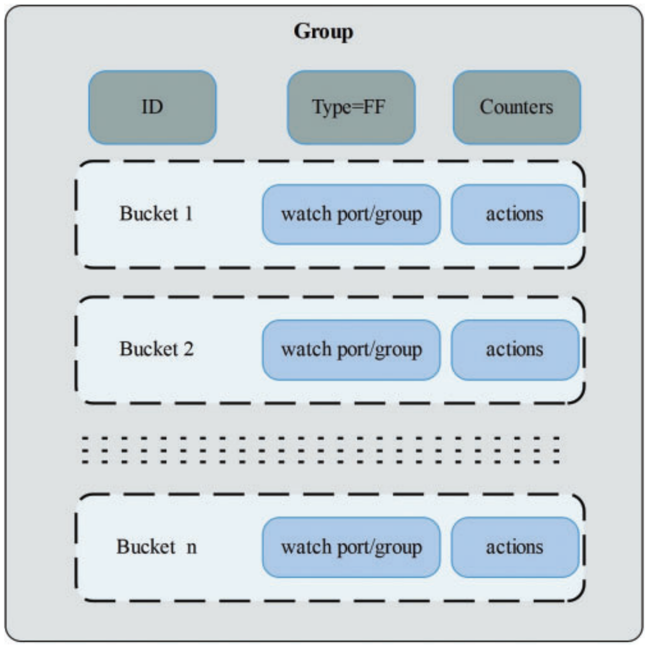

Figure 9: Structure of the group entry

Moreover, Reference [17] aimed to meet provider networks' requirements of 50 ms link failure recovery time, which failure restoration failed to accomplish. Failure restoration needs not less than 100 ms to restore from failures. They use per-link BFD for link failure detection and preconfigured working and backup paths by the controller. One limitation of link-based protection is the need to compute multiple paths. Paths must be computed for all of the nodes between the source and destination when using 1:1 link-based protection.

Continuing with proactive measures, Reference [71] proposed a method in which the controller gathers global topology information on a regular basis (via a topology discovery tool). Using this information and Dijkstra's algorithm, the controller proactively configures both an active and backup path by installing flow entries and a Fast Failover group entry in the OpenFlow (OF) switches along the source-destination route. The group entry of the OF switches on an active path defines two action buckets: the first bucket specifies the active path's output port, and the second bucket specifies the backup path's output port. According to the emulation results, the Fast Failover mechanism has an average recovery time of less than 40 ms, which is significantly faster than the fast restoration mechanism, which requires hundreds of milliseconds.

In another approach, Reference [23] concentrated on shortening the link failure recovery time by reducing the link failure detection time. They used active probing to reduce the detection time. Furthermore, to improve scalability and speed up failure recovery, the central controller is removed from the proposed recovery scheme, with group modifications handled locally by probe stations on ToR switches. Each Top of Rack (ToR) switch in the cluster includes a probe station. This probe station transmits a probe to the aggregation switch, which distributes it to all other ToR switches. Upon failure, all remote ToR switches will not receive probe packets related to the failed link (during a predefined monitoring interval) and will modify the group used to communicate with transmitting ToR. The transmitting ToR, on the other hand, does not receive the probe packets flooded by aggregation switch and modifies all of the groups used to communicate with the cluster's other ToR switches.

In a different approach, Reference [72] proposed SPIDER (**S**tateful **P**rogrammable **F**ailure **D**etection and **R**ecovery). In SPIDER, bidirectional heartbeat packets are used to detect link failures. It is based on Openstate [73], in which each switch has a state table that precedes the flow table, and all flow states are stored in it. SPIDER exploited the feature of FF group, besides using MPLS (Multi-Protocol Label Switching) tags to control the forwarding behaviors. In SPIDER, backup paths are pre-installed in the data plane, enabling the network to recover from link failures without involving the controller.

Further exploring failure recovery, Reference [67] proposed SafeGuard for improving failure recovery in SD-WANs by addressing congestion and switch memory utilization issues. It formulates failure recovery as a multi-objective MILP (Mixed-integer linear programming) optimization problem to efficiently allocate bandwidth and switch memory across any network topology. The solution employs a heuristic for practical computation of backup routes, ensuring network resilience without the need for controller intervention during failures.

In [74], the authors introduced an innovative Fast Failover-Fault Recovery Mechanism in Software Defined Networks (FF-FRM-SDN), designed to enhance network resilience by proactively detecting and recovering from failures through a centralized control mechanism. Employing an Advanced Fast Failover Fault Detection Method (FF_FDM), the system dynamically configures OpenFlow switches and establishes backup paths for rapid failover, significantly reducing recovery times and packet loss. The methodology leverages group entries and flow tables to manage traffic rerouting automatically, optimizing network performance in response to node and link failures. This approach is tested against key performance metrics such as packet loss ratio, end-to-end delay, and failure recovery time, demonstrating substantial improvements over traditional fault management techniques.

Fast Failover Group with flow aggregation

Managing memory resources is a significant challenge in failover protection because additional rules need to be deployed in switches. However, memory in switches, particularly in in those that are TCAM-based switches, is often both limited and costly. To tackle this issue, Reference [75] focused on the size limitation of TCAM memory by introducing a flow table compression algorithm. This method reduces the number of flow entries required for backup paths by aggregating flows that share the same action in of the flow entry, optimizing memory usage without affecting recovery speed. To improve the compression ratio, the authors combined this approach with Plinko [76], a forwarding model that allows the same action to be applied to every packet.

In a related study, Reference [77] proposed a strategy called DFRS (A declarative failure recovery system) to find alternate paths satisfying the flow-specific goals for protection against single link failure. The main goal is to achieve the flow requirement regarding delay and the network's requirement regarding the capacity of the switches' memory. They proposed two algorithms to find the set of backup rules that achieve both requirements. The results indicate that, on average, DFRS requires significantly less memory space, often by an order of magnitude or more, compared to traditional failure protection methods. They used 1) the number of flow entries required and 2) the number of flow entries required for each switch as metrics.

Reference [78] proposed two proactive solutions: one independent on the controller (CIP) and the other is dependent (CDP). The difference between them is that the first one leverages FF group tables and the other depends on Indirect group tables. FF grouping is optional in OpenFlow which causes some vendor switches do not implement this feature in their switches while the Indirect grouping is mandatory in OpenFlow and it is compulsory for vendors to implement this feature in their switches. For flow aggregation they used VLAN tagging to group alternate flows in one flow.

Finally, Reference [21] introduced a novel failure recovery mechanism for software-defined networks (SDNs), utilizing VLAN IDs to expedite the rerouting process during link failures. This approach leverages the Fast Failover group feature in OpenFlow switches, allowing for immediate traffic redirection without controller intervention. The mechanism significantly outperforms traditional recovery methods, achieving recovery times as low as 1.02–1.26 ms and maintaining packet loss rates below 0.28%.

Source routing

Reference [79] proposed a source routing approach in which the controller only installs flow entries at the ingress switches, which reduces the number of flow entries required. The ingress switches encoded the path, expressed by the sequence of egress interfaces at the nodes, in the header of packets using tags. For link failure recovery, the controller can install flow rules to change the tag when a link fails. They used the percentage of flow entries reduction as a metric.

In [80], aimed to efficiently utilize the switches' TCAM usage. It adopts source routing to reduce the number of flow entries in intermediate switches. By encoding the entire path in the packet header, only the ingress switches require flow entries to be installed. In the event of a failure, ingress switches manipulate the path encoding to reroute traffic. Although this method efficiently uses TCAM, it may encounter scalability issues because of the increased header overhead.

Similarly, Reference [81] adopted the concept of source routing. Due to a failure, the ingress switches will write the backup route information to the header of packets, considering avoiding the congestion of links. In another approach, Reference [82] proposed a smart routing approach as a failure recovery framework. Smart routing depends on the historical data of forewarning messages that the controller will exploit to configure backup paths that protect the network from expected failures.

Multiple flow entries with different priorities

Reference [83] utilized different flow entries with different priorities to recover from failures without needing to return to the controller and without requiring the controller to be stateful for OpenFlow ethernet networks. They proposed a new auto-rejection mechanism to evict any flow related to the egress or ingress ports of the switches associated with the failed link. In addition to the auto-rejection mechanism, a set of backup entries should be installed to recover from failure in the least time. They also propose a backup flow entries renewal mechanism by periodically sending a renewal packet to avoid removing backup flow entries. When the network is recovered from the failure, the switches request the working paths that were removed to efficiently utilize resources. They extend the OpenFlow protocol, OpenFlow switch, and OpenFlow controller. They measured performance using 1) the number of flows and 2) switchover time (recovery time) as metrics. The results show that the recovery time is within 64 ms with a high number of entries.

In a similar approach, the main idea in [84] is to install main and backup paths for each flow with different priorities. If the link goes down, the affected switch will inform the controller about the failure, and the affected switch will switch the traffic locally to the backup path after the controller removes all the flow entries related to the failed main path. After the controller removes the affected flow entries related to the failed link, the flow of packets takes the backup path stored in the network. During that, the controller will recompute, using the shortest path algorithm, new best main and backup paths and install them to switches. The packets are then forwarded by the new main path.

Similarly, Reference [85] proposed an approach that installs two paths for each flow in the network by installing two flow entries in each switch along the path. One is the main path, and the other is a backup. When a link failure occurs, the packets related to the affected flows are redirected to the backup path. It is clear that this approach is not suitable for large-scale SDN networks because of the limitations in TCAM memory at switches.

Table 4: Comparison between Protection-based algorithms

Ref.	Aim	Detection	Methodology	Performance metrics	Limitations
[70]	Address scalability and efficiency issues in OFDP to achieve the 50 ms recovery time required by transport networks.	BFD	FF group	1. Recovery time 2. Overhead of using OFDP	1.The testbed is somewhat small. 2. The proposed approach doesn't comply to Openflow protocol.
[79]	To protect flows with efficiently using TCAM of the switches.	LoS	Source routing	Percentage of flow entries reduction	The proposed method adds extra load to packet headers and thus scalability issues in large networks.
[83]	To reduce the recovery time using protection.	LoS	Multiple flow entries with different priorities	1. Recovery time 2. Number of flow entries	Overwhelms the memory of switches which reduce the scalability.
[22]	Integrate large-scale carrier-grade networks with OpenFlow protocol to ensure link failure recovery in under 50 ms.	BFD	FF group	1. Flow-mod transmission capacity 2. Transmitted traffic 3. Flow-mod traffic 4. Recovery time	1. The approach doesn't work for in-band networks. 2. It doesn't consider control channel failures.
[17]	To meet provider 50 ms link failure recovery time.	BFD	FF group	1. Recovery time 2. Number of flow entries	Overwhelms the memory of switches.
[80]	To protect flows with efficiently using TCAM of the switches.	LoS	Source routing	1. Recovery time 2. Number of flow entries	The proposed method adds extra load to packet headers and thus scalability issues in large networks.

(Continued)

Table 4 (continued)

Ref.	Aim	Detection	Methodology	Performance metrics	Limitations
[85]	To meet provider networks requirements of 50 ms link failure recovery time.	LoS	Multiple flow entries with different priorities	Recovery time	Overwhelms the memory of switches which reduce the scalability.
[71]	To reduce detection and recovery time	LoS	FF group	1. Number of flow entries required 2. Number of flow entries required per switch	Depending on continuous monitoring of the data plane should introduce a considerable load on the controller.
[75]	To recover from multiple failures by installing multiple backup paths.	LoS	FF Group with flow aggregation	Recovery time	Increasing the processing load on the controller and overwhelms the memory of switches.
[86]	To protect flows with efficiently using TCAM of the switches.	LoS	Segment routing with flow aggregation	1. Number of flow entries 2. Length of the backup path	Could introduce congestion after recovery.
[72]	To reduce the detection and recovery time.	BFD	FF group	3. Recovery time	Overwhelms the memory of switches and could introduce congestion to the dataplane.
[81]	To develop a low-overhead, congestion-aware, rapid link failure recovery method in SDNs.	LoS	Source routing	1. Recovery time 2. Number of flow entries	1. Extra load to packet headers and thus scalability issues in large networks. 2. Inaccuracies because of stale topology in planning and selecting backup paths.

(Continued)

Table 4 (continued)

Ref.	Aim	Detection	Methodology	Performance metrics	Limitations
[82]	Enhances the availability time of services.	LoS	Source routing	1. Recovery time 2. Number of flow entries	Introduce congestion after recovery.
[67]	To develop a system that enhances failure recovery in SD-WANs by optimizing bandwidth allocation and switch-memory efficiency.	LoS	FF group	1. Number of congested links 2. Length of backup routes 3. Overall link utilization in the network	1. Potential challenges in scaling the heuristic for very large networks due to its NP-hard nature. 2. The need for empirical tuning of the optimization model's parameters.
[21]	To protect flows with efficiently using TCAM of the switches.	LoS	FF group and flow aggregation	1. Recovery time 2. Packet loss rate	1. No evaluation is introduced for Flow entry reduction. 2. The scale of experiments is too small.

Segment routing with flow aggregation

Reference [86] introduced a routing approach designed to minimize TCAM usage. They implemented two routing strategies: Backward Local Rerouting (BLR) and Forward Local Rerouting (FLR). In BLR, for each established flow, a node-disjoint path is calculated, and in the event of a failure along the working path, packets are sent back to their source and rerouted through the backup path. FLR uses a two-phase algorithm. Initially, FLR identifies the backup path for each link in the working path that requires the fewest additional nodes. Next, it resolves rule conflicts (such as different match fields with the same output port) by selecting the rule that is compatible with both paths and discarding the others. To further reduce TCAM consumption, instead of using a Fast Failover group type, the OpenFlow protocol was modified to include a smaller entry in the action set called BACKUP_OUTPUT, which specifies an output port for failures. This modification reduces the number of additional flow entries and switches needed for the backup path compared to the working path.

Similarly, Reference [87] proposed a segment routing model utilizing MPLS labels to manage backup paths efficiently. The method adopts the segmenting of the paths and aggregating affected flows in order to reduce the number of backup paths and address hardware limitations like the

Maximum SID (Segment Identifiers) Depth (MSD). This approach optimizes TCAM usage while maintaining rapid recovery from failures.

Cranckback routing with MPLS tagging

References [88] and [89] presented a protection scheme that relies significantly on MPLS crankback routing. This approach involves using the same data packets, which are initially tagged (for instance, with an MPLS label that carries information about the failure event) and then sent back along the working path. Notably, only the initial packets of the flow are returned from the node that detects the failure. Once the first tagged packet is handled by the reroute node, a state transition occurs in the OpenState switch, and all subsequent packets from the source node are redirected by the reroute node onto the detour.

4.3 Hybrid Methodology

Many of the studies in the literature adopt both restoration and protection methodologies in their proposed approaches. Table 5 presents comparison between hybrid approaches. Indeed, hybrid approaches try to leverage the strengths of both: restoration and protection methodologies in terms of fast failure recovery and efficient TCAM usage. Reference [90] classified the flows into bronze, silver, and gold classes based on their significance. The mechanism to recover from failures could be protection or restoration based on the class. The results show an average recovery time of 30, 40, and 50 ms for bronze, silver, and gold classes, respectively.

In another study, Reference [91] also explored methods to reduce the size of TCAM memory used for link failure protection and minimize link recovery time. Their approach involved using precomputed backup paths, where the entrance switch labels affected packets for forwarding through action buckets in the Fast Failover (FF) group table. This method addresses two main challenges: the number of flow table updates and the number of flow entries required for failure recovery. However, this solution has some drawbacks, including the need for extensive processing at the controller and the frequent installation, deletion, or modification of flow entries.

Similarly, Reference [92] proposed a proactive restoration technique aimed at minimizing service disruptions caused by network failures. Unlike traditional fault management strategies that react to failures after they occur, this approach leverages the predictive capabilities inherent in SDN to pre-emptively reconfigure network routes, thereby preserving service continuity. Experimental results across both real-world and synthetic topologies demonstrate the method's efficacy, achieving service availability improvements of up to 97%.

Table 5: Comparison between hybrid approaches

Reference	Aim	Detection	Method	Performance metrics	Limitation
[92]	To efficiently use the resources in terms of TCAM memory at switches and to recover in a timely manner for time-sensitive traffic	LoS	Classifying the flows based on significance and using protection or restoration based on the significance of the flow	1. Recovery time	1. Needs too much processing at the controller 2. Classification of traffic may be inaccurate

(Continued)

Table 5 (continued)

Reference	Aim	Detection	Method	Performance metrics	Limitation
[91]	To reduce the size of TCAM memory used for link failure protection and the link recovery time	LoS	Using precomputed backup path, with the entrance switch labeling affected packets for forwarding through action buckets in the FF group table	1. Number of flow table updates 2. Number of Flow entries for failure recovery	1. Needs too much processing at the controller 2. Too many flow entry installation/deletion or modifications
[93]	To reduce the size of TCAM memory used for link failure protection and the link recovery time	LoS	The FFRLI scheme employs a hierarchical approach to fault recovery in SDNs by categorizing network links based on their importance, and tailoring proactive, reactive, and hybrid recovery strategies to optimize resource use and minimize recovery times	1. Recovery time 2. TCAM usage 3. Bandwidth utilization	1. Needs too much processing at the controller 2. Classification of links may be inaccurate

In a different approach, Reference [93] presented the Fast Fault Recovery Scheme Based on Link Importance (FFRLI) for SDNs. The method uses a two-stage process that classifies network links by importance, enabling tailored recovery strategies that optimize resource usage and minimize recovery times. By categorizing links as main, minor, or edge, FFRLI prioritizes recovery efforts for critical links while conserving resources for less crucial ones. Simulations in Internet2 and Abilene networks show that FFRLI outperforms traditional recovery methods, significantly reducing recovery times and improving resource efficiency.

4.4 Comparison between Restoration and Protection Mechanisms

Both of restoration and protection have own advantages and drawbacks. In this subsection, we will highlight them and compare between the restoration and protection mechanisms based on the following points (as shown in Table 6):

Table 6: Comparison between restoration and protection approaches

	Restoration	Protection
TCAM consumption	Low	High
Processing overhead the controller	High	Low
Processing overhead at the switches	Low (No flow entries used for recovery)	High (Many flow entries used for recovery)
Recovery time delay	Large	Small
Flexibility	Flexible with topology changes	Static
Congestion awareness	Aware about congestion because its flexibility	Not aware about congestion because of the static configuration
Scalability	Scalable for large networks	Not scalable for large networks particularly for fine-grained flow recovery
CGN requirements	Does not meet the CGN requirements of less than 50 ms of recovery time	Meet the CGN requirements of less than 50 ms of recovery time

1) TCAM size of the switches

Flow entries in SDN switches are stored in Ternary Content Addressable Memory (TCAM), which is small in size, expensive, and energy intensive. Thus, TCAM can only hold a few thousand entries. According to [94], to scale up with demand, servers have to handle hundreds of thousands of clients simultaneously. In the protection approach, a backup path for the failed link must be configured for each flow that traverses the failed link, the matter that introduces scalability issues because of the large number of flow entries that are required to be proactively installed. While in restoration, no backup flow entries are installed prior to the occurrence of link failure. As a result, restoration is more scalable than protection in terms of TCAM size in switches.

2) Processing overhead the controller

In protection, for every new flow, the controller should calculate a backup path and install the required flow rules in the switches along the path. This is impractical in large SDN networks and can cause the controller to be overwhelmed with such computation. Further, the controller should maintain the state of huge number of flows which is also add a considerable burden to the controller processing resources. Restoration also can introduce a considerable overload to the controller, dynamic detouring a large number of the disrupted flows in a short time may overburden the controller and cause performance degradation.

3) Processing overhead at the switches

The protection will induce additional processing for matching the additional flows for the alternate paths.

4) Recovery time delay

In the restoration scheme, the number of flow setup messages exchanged between OpenFlow switches and a controller usually influences the recovery time. The underlying idea behind link failure recovery is to migrate affected flow entries from a failed port to another active port. This migration can be carried out by changing the output port number of the affected flow entries to another port number. In OpenFlow networks, port number changes are typically accomplished by sequentially sending flow setup messages from a controller to a switch. Because message transport is performed sequentially, a larger number of messages requires more recovery time. Moreover, failure restoration may extend recovery time; additionally, the time spent transmitting the flow modification message for per-flow detouring increases as the number of flows to recover grows. Higher recovery times also cause traffic delays and packet loss. As a result, this approach could fail to meet the CGN's delay bounds of 50 ms. While in protection, the backup paths are preconfigured, so if a link fails, the switch can locally redirect the disrupted flows without consulting the controller, reducing recovery time and potentially meeting the CGN's needs.

5) Flexibility

Restoration is considered to be dynamic and flexible by allocating backup paths dynamically because it depends on a consistent network topology. This has a considerable effect on the overall SDN efficiency. For example, how flexibility could affect the SDN networks is link congestion. The configuration of backup paths in protection is static and this may lead to congestion in other links after the failure. While in restoration the controller will consider the utilization of the links while computing the backup path to avoid link congestion.

6) Compatibility

The restoration technique uses the fundamental features in OpenFlow, while protection could use advanced features that are not supported in all OpenFlow versions that are implemented in commodity switches.

7) Scalability

Regarding recovery time, restoration may be suitable in small-scale networks and services that can tolerate with relatively longer delay in case of link failure, but it may not be suitable for real-time services. On the other hand, protection is not appropriate for large-scale networks when considering the overhead on TCAM, as it involves configuring a significant number of flow entries for backup paths for each working path.

5 Research Challenges and Future Directions

This section discusses the challenges in the research area of the link failure recovery that are still unaddressed and proposes future research directions to solve them.

5.1 Research Challenges

Despite existing studies proposing solutions for various problems, there are still several unaddressed issues:

1. Scalability

As networks grow in size and complexity, scalability becomes a critical challenge for both controller and switches resources. In large-scale SDN networks, the number of flow entries required can exceed the capacity of TCAM, leading to a reduction in the number of entries that can be stored or an increase in the number of packets that must be forwarded to the controller for processing. This can result in increased latency, packet loss, and reduced network performance. In restoration-based techniques, frequent path computation can put an additional processing overhead on the controller, and frequently updating the flow entries at the switches can cause bottlenecks.

2. Latency

Meeting the carrier-grade network requirement of less than 50 ms recovery time is challenging especially for large-scale networks. Restoration methods may introduce delays due to the time needed for the controller to detect, compute and install new paths.

3. Security

SDN introduces new security vulnerabilities due to its centralized control architecture. Attackers may exploit these weaknesses in falsifying the controller by causing false failure alarms or by fabricate fake links between switches. The falsified topology at the controller will affect the recovery from link failures.

4. Resource Constraints

Limited processing capacity in controllers and limited memory in switches introduce availability and scalability challenges. Efficient utilization of these resources is fundamental to prevent bottlenecks and ensure timely recovery. Restoration recovery techniques are often preferred for their flexibility in dealing with frequent traffic changes. However, frequent path computation can add additional processing overhead on the controller, and continuous updates to the flow entries at the switches can cause bottlenecks. Conversely, the use of TCAM in SDN networks can be a significant challenge due to its limited size and high cost. Protection-based mechanisms often require a large number of pre-installed flow entries, consuming significant TCAM memory in switches, which is limited and expensive.

5. Congestion Awareness

Post-failure traffic rerouting may lead to congestion on alternative paths. Existing recovery mechanisms, particularly protection-based, may not consider the current network load, potentially causing link congestion and then overall network performance degradation.

5.2 Future Directions

To address the above challenges, we propose the following future research directions:

1. Integration of AI and Machine Learning

Applying AI and machine learning techniques can enhance failure detection and recovery by learning from historical data [95]. These models can predict failures and optimize resource allocation by recommending the type of failure recovery method [96,97].

2. Development of Secure Detection Protocols

Implementing secure protocols for topology discovery and failure detection with authentication and encryption can prevent malicious interference in failure detection and recovery processes by securing the controller from falsified topology [98].

3. Hybrid Recovery Mechanisms

Designing adaptive mechanisms that combine restoration and protection methodology can leverage the advantages of both approaches. Flows can be classified based on QoS requirements which help apply the appropriate recovery method to achieve scalability and efficiency [99].

4. Resource-Efficient Algorithms

Regarding the switches' memory resources, it is recommended to utilize flow aggregation or compression, dynamically managing flow tables, and using hybrid memory architectures. For controller processing capacity, the computational path time, while others involve path selection methods that use a fixed number of operations [100].

5. Congestion-Aware Recovery Strategies

Incorporating real-time traffic monitoring applications into recovery decisions can prevent congestion and ensure smooth traffic flow after failures [101].

6 Conclusion

This survey explored the vital aspects of failure detection and recovery in Software-Defined Networking (SDN), highlighting its promise as a paradigm for centralized control and improved network management. As SDN becomes increasingly important to Carrier Grade networks, the challenge of sustaining high service availability, particularly in detecting and recovering from link failures in the data plane, has become more dominant. A comprehensive survey of the literature exposes a variety of approaches, each with both advantages and disadvantages in terms of efficiency and performance. Restoration-based and protection-based approaches were critically evaluated, providing insights into their respective trade-offs in terms of recovery speed, switches' TCAM memory efficiency, and implementation complexity.

Despite the existence of different significant approaches, the survey highlights persistent gaps that are required to be addressed to realize the SDN in modern network infrastructures. Future research must focus on developing more adaptive, scalable, and efficient failure recovery techniques that can meet the requirements of next-generation networks. This includes refining detection algorithms for faster response times, optimizing recovery methods to minimize the recovery time and efficiently use limited resources such as the memory of the switches at the data plane, and exploring hybrid approaches that utilize the strengths of both restoration and protection approaches. Additionally, integrating AI and machine learning and enhancing security will be crucial. Considering and addressing these challenges, will help to speed up the realization of SDN at the scale of modern networks, ensuring service reliability. Moreover, improving link failure detection and recovery mechanisms has significant implications for network performance, reducing downtime, enhancing reliability, and supporting real-world applications such as cloud services, Internet of Things (IoT), and 5G networks.

Acknowledgement: The anonymous reviewers of Computers, Materials & Continua journal are highly acknowledged for their valuable comments, which enhanced the quality of this paper.

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

Author Contributions: Suheib Alhiyari was responsible for ideation, research, development, and writing of the initial draft. Siti Hafizah AB Hamid and Nur Nasuha Daud reviewed the manuscript and provided comments to enhance the first version. They also 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 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 no conflicts of interest to report regarding the present study.

References

[1] J. Ali, G. Lee, B. 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.

[2] W. Rafique, L. Qi, I. Yaqoob, M. Imran, R. U. Rasool and W. Dou, "Complementing IoT services through software defined networking and edge computing: A comprehensive survey," *IEEE Commun. Surv. Tutorials.*, vol. 22, no. 3, pp. 1761–1804, 2020. doi: 10.1109/COMST.2020.2997475.

[3] D. Sanvito and A. Geraci, "Traffic management in networks with programmable data planes," 2021. Accessed: Nov. 15, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:226000493

[4] T. Semong *et al.*, "A review on Software Defined Networking as a solution to link failures," *Sci. African*, vol. 21, Feb. 2023, Art. no. e01865. doi: 10.1016/j.sciaf.2023.e01865.

[5] N. Khan, R. Bin Salleh , A. Koubaa, Z. Khan, M. K. Khan and I. Ali, "Data plane failure and its recovery techniques in SDN: A systematic literature review," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 35, no. 3, pp. 176–201, 2023. doi: 10.1016/j.jksuci.2023.02.001.

[6] D. Turner, K. Levchenko, A. C. Snoeren, and S. Savage, "California fault lines: Understanding the causes and impact of network failures," *Comput. Commun. Rev.*, vol. 40, no. 4, pp. 315–326, 2010. doi: 10.1145/1851275.1851220.

[7] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C. -N. Chuah, and C. Diot, "Characterization of failures in an IP backbone," in *IEEE INFOCOM 2004*, IEEE, 2004, pp. 2307–2317. doi: 10.1109/INFCOM.2004.1354653.

[8] R. Govindan, I. Minei, M. Kallahalla, B. Koley, and A. Vahdat, "Evolve or die: High-availability design principles drawn from Google's network infrastructure," in *SIGCOMM 2016-Proc. 2016 ACM Conf. Spec. Interes. Gr. Data Commun.*, 2016, pp. 58–72. doi: 10.1145/2934872.2934891.

[9] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "In-band control, queuing, and failure recovery functionalities for openflow," *IEEE Netw.*, vol. 30, no. 1, pp. 106–112, 2016. doi: 10.1109/MNET.2016.7389839.

[10] P. Thorat, S. M. Raza, D. S. Kim, and H. Choo, "Rapid recovery from link failures in software-defined networks," *J. Commun. Netw.*, vol. 19, no. 6, pp. 648–665, 2017. doi: 10.1109/JCN.2017.000105.

[11] ITU-T, "G.1010: End-user multimedia QoS categories," *Int. Telecommun. Union* 1010, 2001. Accessed: Nov. 15, 2024. [Online]. Available: http://scholar.google.com.au/scholar?hl=en&q=ITU-T+Recommendation+G.1010&btnG=&as_sdt=1,5&as_sdtp=#7

[12] S. Ahmad and A. H. Mir, *Scalability, Consistency, Reliability and Security in SDN Controllers: A Survey of Diverse SDN Controllers*. USA: Springer, 2021, vol. 29. doi: 10.1007/s10922-020-09575-4.

[13] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, Apr. 2014. doi: 10.1145/2602204.2602219.

[14] D. Kreutz, F. M. V. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proc. Second ACM SIGCOMM Workshop Hot Top. Softw. Defin. Netw.-HotSDN'13*, New York, NY, USA, ACM Press, 2013. doi: 10.1145/2491185.2491199.

[15] Z. A. Bhuiyan, S. Islam, M. M. Islam, A. B. M. A. Ullah, F. Naz and M. S. Rahman, "On the (in)security of the control plane of SDN architecture: A survey," *IEEE Access*, vol. 11, pp. 91550–91582, 2023. doi: 10.1109/ACCESS.2023.3307467.

[16] S. Shirali-Shahreza and Y. Ganjali, "Efficient implementation of security applications in openflow controller with FleXam," in *2013 IEEE 21st Ann. Symp. High-Perform. Interconn.*, IEEE, Aug. 2013, pp. 49–54. doi: 10.1109/HOTI.2013.17.

[17] N. L. M. Van Adrichem, B. J. Van Asten, and F. A. Kuipers, "Fast recovery in software-defined networks," in *Proc. 2014 3rd Eur. Work. Software-Defined Networks, EWSDN 2014*, 2014, pp. 61–66. doi: 10.1109/EWSDN.2014.13.

[18] Open Networking Foundation, "SDN architecture: Overview and principles." Accessed: Jul. 1, 2023. [Online]. Available: https://opennetworking.org.

[19] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Enabling fast failure recovery in OpenFlow networks," in *2011 8th Int. Workshop Des. Reliable Commun. Netw. (DRCN)*, Krakow, Poland, IEEE, 2011, pp. 164–171. doi: 10.1109/DRCN.2011.6076899.

[20] 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, 2017. doi: 10.1007/s11277-017-4618-0.

[21] H. Nurwarsito and G. Prasetyo, "Implementation failure recovery mechanism using VLAN ID in software defined networks," *Int. J. Adv. Comput. Sci. Appl.*, vol. 14, no. 1, pp. 709–714, 2023. doi: 10.14569/issn.2156-5570.

[22] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "OpenFlow: Meeting carrier-grade recovery requirements," *Comput. Commun.*, vol. 36, no. 6, pp. 656–665, 2013. doi: 10.1016/j.comcom.2012.09.011.

[23] B. Raeisi and A. Giorgetti, "Software-based fast failure recovery in load balanced SDN-based datacenter networks," in *Proc. 6th Int. Conf. Inf. Commun. Manag. (ICICM)*, 2016, pp. 95–99. doi: 10.1109/INFOCOMAN.2016.7784222.

[24] B. Isyaku, K. Bin Abu Bakar, M. N. Yusuf, and M. S. Mohd Zahid, "Software defined networking failure recovery with flow table aware and flows classification," in *ISCAIE 2021-IEEE 11th Symp. Comput. Appl. Ind. Electron.*, 2021, pp. 337–342. doi: 10.1109/ISCAIE51753.2021.9431786.

[25] A. Vishnoi, R. Poddar, V. Mann, and S. Bhattacharya, "Effective switch memory management in OpenFlow networks," in *DEBS 2014-Proc. 8th ACM Int. Conf. Distrib. Event-Based Syst.*, 2014, pp. 177–188. doi: 10.1145/2611286.2611301.

[26] Z. Liu, M. Lian, J. Guo, and G. Wang, "An efficient flow detection and scheduling method in data center networks," in *CONF-CDS 2021: The 2nd Int. Conf. Comput. Data Sci.*, pp. 1–7, 2021. doi: 10.1145/3448734.3450819.

[27] T. Zhang, Y. Lei, Q. Zhang, S. Zou, J. Huang and F. Li, "Fine-grained load balancing with traffic-aware rerouting in datacenter networks," *J. Cloud Comput.*, vol. 10, no. 1, 2021. doi: 10.1186/s13677-021-00252-8.

[28] Y. Al Mtawa, A. Haque, and H. Lutfiyya, "Migrating from legacy to software defined networks: A network reliability perspective," *IEEE Trans. Reliab.*, vol. 70, no. 4, pp. 1525–1541, Dec. 2021. doi: 10.1109/TR.2021.3066526.

[29] J. Chen, J. Chen, F. Xu, M. Yin, and W. Zhang, "When software defined networks meet fault tolerance: A survey," in *Lecture Notes in Computer Science*, Cham: Springer International Publishing, 2015, vol. 9530, pp. 351–368, doi: 10.1007/978-3-319-27137-8_27.

[30] P. C. Fonseca and E. S. Mota, "A survey on fault management in software-defined networks," *IEEE Commun. Surv. Tutori.*, vol. 19, no. 4, pp. 2284–2321, 2017. doi: 10.1109/COMST.2017.2719862.

[31] A. U. Rehman, R. L. Aguiar, and J. P. Barraca, "Fault-tolerance in the scope of Software-Defined Networking (SDN)," *IEEE Access*, vol. 7, pp. 124474–124490, 2019. doi: 10.1109/ACCESS.2019.2939115.

[32] F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska, "Efficient topology discovery in OpenFlow-based Software Defined Networks," *Comput. Commun.*, vol. 77, no. Supplement C, pp. 52–61, 2016. doi: 10.1016/j.comcom.2015.09.013.

[33] S. Khan, A. Gani, A. W. A. Wahab, M. Guizani, and M. K. Khan, "Topology discovery in software defined networks: Threats, taxonomy, and state-of-the-art," *IEEE Commun. Surv. Tutori.*, vol. 19, no. 1, pp. 303–324, 2017. doi: 10.1109/COMST.2016.2597193.

[34] A. Azzouni, R. Boutaba, N. T. M. Trang, and G. Pujolle, "sOFTDP: Secure and efficient OpenFlow topology discovery protocol," in *NOMS 2018-2018 IEEE/IFIP Netw. Operat. Manag. Symp.*, IEEE, Apr. 2018, pp. 1–7. doi: 10.1109/NOMS.2018.8406229.
[35] Linux Foundation, "OpenDaylight," 2019. Accessed: Nov. 15, 2023. [Online]. Available: https://www.opendaylight.org/
[36] Project Floodlight Team, "Project Floodlight," Accessed: Jan. 2, 2024. [Online]. Available: https://floodlight.atlassian.net/wiki/display/floodlightcontroller/Supported+Topologies
[37] R. Jawaharan, P. M. Mohan, T. Das, and M. Gurusamy, "Empirical evaluation of SDN controllers using mininet/wireshark and comparison with cbench," in *2018 27th Int. Conf. Comput. Commun. Netw. (ICCCN)*, IEEE, Jul. 2018, pp. 1–2. doi: 10.1109/ICCCN.2018.8487382.
[38] Ryu Development Team, "Ryu SDN controller," Accessed: Jan. 2, 2024. [Online]. Available: https://ryu-sdn.org/
[39] D. Erickson, "The beacon openflow controller," in *Proc. Second ACM SIGCOMM Workshop Hot Top. Softw. Defin. Netw.-HotSDN'13*, New York, New York, USA, ACM Press, 2013, p. 13. doi: 10.1145/2491185.2491189.
[40] "Cisco open SDN controller," 2019. Accessed: Nov. 15, 2024. [Online]. Available: http://www.cisco.com/c/en/us/products/cloud-systemsmanagement/opensdncontroller/index.html
[41] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning network visibility in software-defined networks: New attacks and countermeasures," in *Proc. 2015 Netw. Distrib. Syst. Secur. Symp.*, Reston, VA, Internet Society, 2015. doi: 10.14722/ndss.2015.23283.
[42] X. Zhao, L. Yao, and G. Wu, "ESLD: An efficient and secure link discovery scheme for software-defined networking," *Int. J. Commun. Syst.*, vol. 31, no. 10, Jul. 2018, Art. no. e3552. doi: 10.1002/dac.3552.
[43] GENI Project Team, "OpenFlow Discovery Protocol," 2021. Accessed: Jan. 2, 2024. Available: http://groups.geni.net/geni/wiki/OpenFlowDiscoveryProtocol
[44] A. Mayoral, R. Vilalta, R. Muñoz, R. Casellas, and R. Martínez, "SDN orchestration architectures and their integration with Cloud Computing applications," *Opt. Switch. Netw.*, vol. 26, no. 10, pp. 2–13, 2017. doi: 10.1016/j.osn.2015.09.007.
[45] Y. Jia, L. Xu, Y. Yang, and X. Zhang, "Lightweight automatic discovery protocol for openflow-based software defined networking," *IEEE Commun. Lett.*, vol. 24, no. 2, pp. 312–315, Feb. 2020. doi: 10.1109/LCOMM.2019.2956033.
[46] L. Ochoa Aday, C. Cervelló Pastor, and A. ernández Fernández, "Current trends of topology discovery in OpenFlow-based software defined networks," 2015. doi: 10.13140/RG.2.2.12222.89929.
[47] L. Ochoa-Aday, C. Cervello-Pastor, A. Fernandez-Fernandez, "eTDP: Enhanced topology discovery protocol for software-defined networks," *IEEE Access*, vol. 7, pp. 23471–23487, 2019. doi: 10.1109/ACCESS.2019.2899653.
[48] M. T. BAH, A. Azzouni, M. T. Nguyen, and G. Pujolle, "Topology discovery performance evaluation of opendaylight and ONOS controllers," in *2019 22nd Conf. Innov. Clouds, Inter. Netw. Workshops (ICIN)*, IEEE, Feb. 2019, pp. 285–291. doi: 10.1109/ICIN.2019.8685915.
[49] M. T. Bah, V. Del-Piccolo, M. Bourguiba, and K. Haddadou, "A centralized controller to improve fault tolerance in TRILL-based fabric networks," in *2016 3rd Smart Cloud Netw. Syst. (SCNS)*, IEEE, Dec. 2016, pp. 1–6. doi: 10.1109/SCNS.2016.7870564.
[50] D. Staessens, S. Sharma, D. Colle, M. Pickavet, and P. Demeester, "Software defined networking: Meeting carrier grade requirements," in *2011 18th IEEE Workshop on Local Metropol. Area Netw. (LANMAN)*, IEEE, 2011, pp. 1–6.
[51] Open Networking Foundation (ONF), "OpenFlow specification," Version 1.5.1, Open Networking Foundation, 2015. Accessed: Jan. 2, 2024. [Online]. Available: https://opennetworking.org/openflow/.
[52] D. W. D. Katz, "Bidirectional forwarding detection," 2010. doi: 10.17487/rfc5880.
[53] S. S. W. Lee, K. Y. Li, K. Y. Chan, G. H. Lai, and Y. C. Chung, "Path layout planning and software based fast failure detection in survivable OpenFlow networks," in *10th Int. Conf. Des. Reliab. Commun. Netw.*, 2014, pp. 1–8. doi: 10.1109/DRCN.2014.6816141.

[54] K. Y. Chan, C. H. Chen, Y. H. Chen, Y. J. Tsai, S. S. W. Lee and C. S. Wu, “Fast failure recovery for in-band controlled multi-controller openflow networks,” in *9th Int. Conf. Inf. Commun. Technol. Converg. ICT Converg. Pow. Smart Intell. ICTC 2018*, 2018, pp. 396–401. doi: 10.1109/ICTC.2018.8539715.

[55] U. C. Kozat, G. Liang, and K. Kokten, “On diagnosis of forwarding plane via static forwarding rules in Software Defined Networks,” in *Proc. IEEE INFOCOM*, 2014, pp. 1716–1724. doi: 10.1109/INFOCOM.2014.6848109.

[56] D. Lopez-Pajares, G. Ibanez, J. M. Arco, B. N. Constantin, and E. Rojas, “Combined ARP-Path & RSTP bridges for smooth migration to robust shortest path bridging,” in *2019 IEEE 44th Conf. Local Comput. Netw. (LCN)*, IEEE, Oct. 2019, pp. 93–96. doi: 10.1109/LCN44214.2019.8990689.

[57] K. Gilbert and S. M. Musa, “Open shortest path first protocol with failure recovery in IP network performance measurement,” in *2021 6th IEEE Int. Conf. Recent Adv. Innov. Eng. (ICRAIE)*, IEEE, Dec. 2021, pp. 1–4. doi: 10.1109/ICRAIE52900.2021.9703992.

[58] S. Sharma, D. Colle, and M. Pickavet, “Enabling fast failure recovery in OpenFlow networks using route-Flow,” in *IEEE Work. Local Metrop. Area Networks*, vol. 2020, 2020. doi: 10.1109/LANMAN49260.2020.

[59] J. Nagano and N. Shinomiya, “A failure recovery method based on cycle structure and its verification by OpenFlow,” in *Proc. Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, 2013, pp. 298–303. doi: 10.1109/AINA.2013.81.

[60] J. Li, J. Hyun, J. -H. Yoo, S. Baik, and J. W. -K. Hong, “Scalable failover method for Data Center Networks using OpenFlow,” in *2014 IEEE Netw. Operat. Manag. Symp. (NOMS)*, IEEE, May 2014, pp. 1–6. doi: 10.1109/NOMS.2014.6838393.

[61] S. Astaneh and S. S. Heydari, “Multi-failure restoration with minimal flow operations in software defined networks,” in *2015 11th Int. Conf. Des. Reliab. Commun. Networks, DRCN 2015*, 2015, pp. 263–266. doi: 10.1109/DRCN.2015.7149024.

[62] A. Malik, B. Aziz, M. Adda, and C. H. Ke, “Optimisation methods for fast restoration of software-defined networks,” *IEEE Access*, vol. 5, pp. 16111–16123, 2017. doi: 10.1109/ACCESS.2017.2736949.

[63] X. Zhang, Z. Cheng, R. Lin, L. He, S. Yu and H. Luo, “Local fast reroute with flow aggregation in software defined networks,” *IEEE Commun. Lett.*, vol. 21, no. 4, pp. 785–788, Apr. 2017. doi: 10.1109/LCOMM.2016.2638430.

[64] K. Qiu, J. Zhao, X. Wang, X. Fu, and S. Secci, “Efficient recovery path computation for fast reroute in large-scale software-defined networks,” *IEEE J. Sel. Areas Commun.*, vol. 37, no. 8, pp. 1755–1768, 2019. doi: 10.1109/JSAC.2019.2927098.

[65] D. B. Swarna and V. Muthumanikandan, “Nested failure detection and recovery in software defined networks,” in *Proc. 2019 3rd IEEE Int. Conf. Electr. Comput. Commun. Technol., ICECCT 2019*, 2019, pp. 1–6. doi: 10.1109/ICECCT.2019.8869085.

[66] H. Kim, M. Schlansker, J. R. Santos, J. Tourrilhes, Y. Turner and N. Feamster, “Coronet: Fault tolerance for software defined networks,” in *2012 20th IEEE Int. Conf. on Netw. Proto. (ICNP)*, IEEE, 2012, pp. 1–2.

[67] M. Shojaee, M. Neves, and I. Haque, “SafeGuard: Congestion and memory-aware failure recovery in SD-WAN,” in *2020 16th Int. Conf. Netw. Serv. Manag. (CNSM)*, IEEE, Nov. 2020, pp. 1–7. doi: 10.23919/CNSM50824.2020.9269119.

[68] I. V. Bastos, V. C. Ferreira, D. C. Muchaluat-Saade, C. V. N. De Albuquerque, and I. M. Moraes, “Path recovery algorithm using fast-failover for software-defined networks,” in *2020 4th Conf. Cloud Internet Things, CIoT 2020*, 2020, pp. 49–52. doi: 10.1109/CIoT50422.2020.9244292.

[69] E. Bellagamba, J. Kempf, and P. Sköldström, “Link failure detection and traffic redirection in an openflow network,” vol. 1, no. 19, 2011. Accessed: Jan. 2, 2024, [Online]. Available: http://www.diva-portal.org/smash/record.jsf?pid=diva2:613241.

[70] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takacs and P. Skoldstrom, “Scalable fault management for OpenFlow,” in *2012 IEEE Int. Conf. Commun. (ICC)*, IEEE, Jun. 2012, pp. 6606–6610. doi: 10.1109/ICC.2012.6364688.

[71] Y. D. Lin, H. Y. Teng, C. R. Hsu, C. C. Liao, and Y. C. Lai, “Fast failover and switchover for link failures and congestion in software defined networks,” in *2016 IEEE Int. Conf. Commun., ICC 2016*, 2016. doi: 10.1109/ICC.2016.7510886.

[72] C. Cascone, D. Sanvito, L. Pollini, A. Capone, and B. Sansò, “Fast failure detection and recovery in SDN with stateful data plane,” *Int. J. Netw. Manag.*, vol. 27, no. 2, pp. 1–14, 2017. doi: 10.1002/nem.1957.

[73] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, “OpenState,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 44–51, Apr. 2014. doi: 10.1145/2602204.2602211.

[74] S. S. Kumar, S. Sharathkumar, R. Scholar, and N. Sreenath, “A fast failover technique for link failures and proactive controller based fault recovery mechanism in software deened networks a fast failover technique for link failures and proactive controller based fault recovery mechanism in software defined network,” 2022. doi: 10.21203/rs.3.rs-1975833/v1.

[75] B. Stephens, A. L. Cox, and S. Rixner, “Scalable multi-failure fast failover via forwarding table compression,” in *Proc. Symp. SDN Res.*, New York, NY, USA, ACM, Mar. 2016, pp. 1–12. doi: 10.1145/2890955.2890957.

[76] B. Stephens, A. L. Cox, and S. Rixner, “Plinko: Building provably resilient forwarding tables,” in *Proc. 12th ACM Work. Hot Top. Networks, HotNets 2013*, 2013. doi: 10.1145/2535771.2535774.

[77] H. Li, Q. Li, Y. Jiang, T. Zhang, and L. Wang, “A declarative failure recovery system in software defined networks,” in *2016 IEEE Int. Conf. Commun., ICC 2016*, 2016. doi: 10.1109/ICC.2016.7510887.

[78] P. Thorat, S. Jeon, and H. Choo, “Enhanced local detouring mechanisms for rapid and lightweight failure recovery in OpenFlow networks,” *Comput. Commun.*, vol. 108, pp. 78–93, Aug. 2017. doi: 10.1016/j.comcom.2017.04.005.

[79] M. Soliman, B. Nandy, I. Lambadaris, and P. Ashwood-Smith, “Source routed forwarding with software defined control, considerations and implications,” in *Conex. Student 2012-Proc. ACM Conf. 2012 Conex. Student Work.*, 2012, pp. 43–44. doi: 10.1145/2413247.2413274.

[80] M. Soliman, B. Nandy, I. Lambadaris, and P. Ashwood-Smith, “Exploring source routed forwarding in SDN-based WANs,” in *2014 IEEE Int. Conf. Commun., ICC 2014*, 2014, pp. 3070–3075. doi: 10.1109/ICC.2014.6883792.

[81] L. Huang, Q. Shen, and W. Shao, “Congestion aware fast link failure recovery of SDN network based on source routing,” *KSII Trans. Internet Inf. Syst.*, vol. 11, no. 11, pp. 5200–5222, 2017. doi: 10.3837/tiis.2017.11.002.

[82] A. Malik, B. Aziz, M. Adda, and C. H. Ke, “Smart routing: Towards proactive fault handling of software-defined networks,” *Comput. Networks*, vol. 170, 2020, Art. no. 107104. doi: 10.1016/j.comnet.2020.107104.

[83] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, “OpenFlow-based segment protection in ethernet networks,” *J. Opt. Commun. Netw.*, vol. 5, no. 9, Sep. 2013, Art. no. 1066. doi: 10.1364/JOCN.5.001066.

[84] N. M. Sahri and K. Okamura, “Fast failover mechanism for software defined networking-openflow based,” in *CFI ’14: Proc. Ninth Int. Conf. Future Internet Technologi.*, 2014, pp. 2–3, doi: 10.1145/2619287.2619303.

[85] V. Padma and P. Yogesh, “Proactive failure recovery in OpenFlow based Software Defined Networks,” in *2015 3rd Int. Conf. Signal Process. Commun. Networking, ICSCN 2015*, 2015. doi: 10.1109/ICSCN.2015.7219846.

[86] P. M. Mohan, T. Truong-Huu, and M. Gurusamy, “TCAM-aware local rerouting for fast and efficient failure recovery in software defined networks,” 2016. doi: 10.1109/glocom.2015.7417309.

[87] S. Wang, H. Xu, L. Huang, X. Yang, and J. Liu, “Fast recovery for single link failure with segment routing in SDNs,” in *Proc. 21st IEEE Int. Conf. High Perform. Comput. Commun. 17th IEEE Int. Conf. Smart City 5th IEEE Int. Conf. Data Sci. Syst. HPCC/SmartCity/DSS 2019*, 2019, pp. 2013–2018. doi: 10.1109/HPCC/SmartCity/DSS.2019.00278.

[88] C. Cascone, L. Pollini, D. Sanvito, and A. Capone, “Traffic management applications for stateful SDN data plane,” in *Proc. Eur. Work. Softw. Defin. Networks (EWSDN)*, 2015, pp. 85–90. doi: 10.1109/EWSDN.2015.66.

[89] A. Capone, C. Cascone, A. Q. T. Nguyen, and B. Sanso, "Detour planning for fast and reliable failure recovery in SDN with OpenState," in *2015 11th Int. Conf. Des. Reliab. Commun. Netw. (DRCN)*, IEEE, Mar. 2015, pp. 25–32. doi: 10.1109/DRCN.2015.7148981.

[90] D. Adami, S. Giordano, M. Pagano, and N. Santinelli, "Class-based traffic recovery with load balancing in software-defined networks," in *2014 IEEE Globecom Work., GC Wkshps. 2014*, 2014, pp. 161–165. doi: 10.1109/GLOCOMW.2014.7063424.

[91] 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, Jul. 2020. doi: 10.1109/JSAC.2020.2986668.

[92] A. Malik and R. De Frein, "A proactive-restoration technique for SDNs," in *Proc. IEEE Symp. Comput. Commun.*, 2020, vol. 2020, pp. 1–6. doi: 10.1109/ISCC50000.2020.9219598.

[93] Z. Zhu, H. Yu, Q. Liu, D. Liu, and B. Mei, "FFRLI: Fast fault recovery scheme based on link importance for data plane in SDN," *Comput. Networks*, vol. 237, 2023, Art. no. 110062. doi: 10.1016/j.comnet.2023.110062.

[94] Y. Zhao, A. Saeed, M. Ammar, and E. Zegura, "Scouting the path to a million-client server," vol. 12671, pp. 337–354, 2021. doi: 10.1007/978-3-030-72582-2.

[95] Y. Wang *et al.*, "Virtual network fault management platform and mechanism based on big data," in *2023 IEEE Int. Symp. Broadband Multim. Syst. Broadcast. (BMSB)*, IEEE, Jun. 2023, pp. 1–4. doi: 10.1109/BMSB58369.2023.10211122.

[96] M. Moseva and V. Lipatov, "Research and development of algorithms for improving fault tolerance in SDN networks based on artificial intelligence," in *2024 Wave Electron. Appl. Inform. Telecommun. Syst. (WECONF)*, IEEE, Jun. 2024, pp. 1–5. doi: 10.1109/WECONF61770.2024.10564664.

[97] Y. Dong *et al.*, "Research on fault management system based on artificial intelligence in data network," in *2023 IEEE Int. Symp. Broadband Multim. Syst. Broadcast. (BMSB)*, IEEE, Jun. 2023, pp. 1–5. doi: 10.1109/BMSB58369.2023.10211345.

[98] S. Soltani, A. Amanlou, M. Shojafar, and R. Tafazolli, "Security of topology discovery service in SDN: Vulnerabilities and countermeasures," *IEEE Open J. Commun. Soc.*, vol. 5, pp. 3410–3450, 2024. doi: 10.1109/OJCOMS.2024.3406489.

[99] F. Huicong *et al.*, "SDN network reliability guarantee mechanism based on network characteristics," in *2023 IEEE 7th Inform. Technol. Mechatron. Eng. Conf. (ITOEC)*, IEEE, Sep. 2023, pp. 1093–1097. doi: 10.1109/ITOEC57671.2023.10291757.

[100] X. Zhang and J. Chen, "ATL: A link failure recovery method with fast recovery speed, low interruption rate, and small TCAM consumption in SDN," in *2023 IEEE 29th Int. Conf. Paral. Distrib. Syst. (ICPADS)*, IEEE, Dec. 2023, pp. 2083–2090. doi: 10.1109/ICPADS60453.2023.00283.

[101] G. N. Kumar, K. Katsalis, P. Papadimitriou, P. Pop, and G. Carle, "Failure handling for time-sensitive networks using SDN and source routing," in *2021 IEEE 7th Int. Conf. Netw. Softwariz. (NetSoft)*, IEEE, Jun. 2021, pp. 226–234. doi: 10.1109/NetSoft51509.2021.9492666.