IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 31 October 2022, accepted 11 November 2022, date of publication 17 November 2022,
date of current version 23 November 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3222849

SURVEY

Dynamic Routing and Failure Recovery Approaches for Efficient Resource Utilization in OpenFlow-SDN: A Survey

BABANGIDA ISYAKU[1,2], KAMALRULNIZAM BIN ABU BAKAR[1], (Member, IEEE), FUAD A. GHALEB[1], AND ABDULAZIZ AL-NAHARI[3]

[1]Department of Computer Science, School of Computing, Faculty of Engineering, Universiti Teknologi Malaysia, Johor Bahru, Johor 81310, Malaysia
[2]Department of Computer Science, Sule Lamido University, Kano, Jigawa 700271, Nigeria
[3]UNITAR Graduate School, UNITAR International University, Petaling Jaya 47301, Malaysia

Corresponding authors: Babangida Isyaku (isyaku.babangida@utm.my) and Abdulaziz Al-Nahari (abdulaziz.yahya@unitar.my)

This work was supported in part by the Universiti Teknologi Malaysia under Grant R.J130000.7709.4J561, and in part by the Post-Doctoral Fellowship Scheme under Grant Q.J130000.21A2.06E03.

ABSTRACT Software Defined Networks (SDN) is a new network paradigm that emerged to offer better network management by separating network control logic and data forwarding element. This separation speeds up network innovation without relying on the vendor-proprietary interface for network element configuration to forward packets. However, SDN is flow driven network; for each arrived flow, a feasible path is computed to deliver the flow to its destination. Afterwards, the SDN control logic process the corresponding routing rules and instruct the set of data forwarding elements to install them on their Flowtable to guide the routing process. Unfortunately, the network changes more frequently in dynamic large-scale networks, and the Flowtable is a constraint with limited space. These challenges require the SDN controller to compute paths more often, which may also require many flows routing rules. In addition, the frequency of communication link failures has increased lately. The successful deployment of SDN heavily depends on how it satisfies the reliability requirement with uninterrupted services. Several studies were conducted to compute the optimal path for data forward to meet their Quality-of-Service demand. Other studies focus on reducing the frequency of link failure. Some studies were conducted to manage the constraint Flowtable resources. This survey focuses on Routing rules placement, unoptimized routing, link, and switch load balancing, failure detection, and recovery. The paper extensively discusses each issue and analyses the weakness of the current solutions. Finally, it highlights potential challenges that need future research attention.

INDEX TERMS SDN, OpenFlow, route path selection, load balancing, failure recovery.

I. INTRODUCTION

The rapid growth of data centres and the emergence of the Internet of Things (IoT) have increased the number of network-connected devices. Integrating these heterogeneous devices enables humans to interact easily with their surrounding physical world, boosting business growth. Network traffic control in these modern networks is a very complex task that requires incorporating the dynamicity of time-varying changes in the network environment over a heterogeneous network. Traditionally, network operators constantly used a command line interface using vendor software to configure the network devices. For any subtle network changes, operators struggle too much with the manual reconfiguring processes, which could account for around 40% of the most typical network operational issues [1]. Unfortunately, the architecture is not well designed to enable fine-grained and Quality of Service (QoS) aware traffic engineering over the network. Integrating the network control element with the data forwarding entities complicates the network traffic

The associate editor coordinating the review of this manuscript and approving it for publication was Adamu Murtala Zungeru.

TABLE 1. Comparison of previous survey scope and their contributions.

Related Works	Work Scope	Failure Recovery	Dynamic Routing	Survey Technical Contribution
Fonseca et al., [10]	Present comprehensive fault management and threats survey, focussing on each layer in SDN	√	X	It discussed fault management at each SDN layer and highlighted future research directions
Yu *et al.* [2]	Provide SDN systematic fault management survey. Advancements in both academia and industry were presented	√	X	An in-depth analysis of fault management focussing on failure detection, localizing, correcting, and prevention was discussed
Rehman et al., [11]	Discussed fault tolerance at the control and data plane	√	X	Structure fault tolerance according to SDN layers and provide future research direction
Ali et al, [12]	Investigate the open research question posed due to the SDN centralized architecture	√	X	It summarized failure recovery approaches by analyzing restoration and protection
Malik et al., [3]	Present data plane fault tolerance and recovery. The discussed	√	X	Classify SDN recovery schemes and present future research challenges
Present Paper	Present dynamic routing and failure recovery approaches for resource utilization	√	√	IT extensively discussed the existing routing and failure detection and recovery approaches. It further provides direction directions

monitoring process, leading to less QoS-aware flow control. As a result, inefficient network resource utilization is unavoidable, and network management is quite challenging, hindering network innovation.

Software Defined Networks (SDN) emerged to simplify network management and speed up innovation by separating the control plane from the data plane. SDN moved the network state and intelligence to a logically centralized controller. The data plane becomes a simpler forwarding entity that forwards packets to the desired destination. The controller extracts the high-level network application through the northbound interface and manages the network devices through the southbound protocol [2]. This way, network managers leverage SDN controllers to implement new dynamic routing strategies, customized traffic management, dynamic allocation of network resources, and many other programmable functionalities. An OpenFlow is the most popular protocol that hides the network devices' complexity and exposes a simple Application Programming Interface (API) to the network operators. This way, operators are relieved from the manual configuration of devices and resource management. Thus, accelerating network management and innovation and reducing the risk of an error that may arise during the network manual configuration time. The merit of SDN made it to be considered under the deployment in Carrier-Grade Networks (CGNs). It has attracted the attention of academia and commercial industries, such as Google, Microsoft, Deutsche Telekom, and Verizon [3].

Despite the advantages of separating the network control logic and data forwarding entities introduced by SDN, there have been various concerns about network resource utilization and performance issues. The communication channel between the control logic and data forwarding element introduced extra processing delay and increased the amount of control traffic managed by the network controller. As the network increases, a high traffic volume is exchanged between the two entities. The data forwarding entities also introduced another concern of shortage of storage to accommodate the required traffic flows. QoS flows have stringent routing requirements, which must comply for optimal performance. Fault tolerance is a crucial property for computer network availability. Unfortunately, an increased communication link failure at the SDN data plane was recently reported.

Many studies have been proposed to compute feasible paths while meeting the demand of flow QoS and preserving the network resource [4], [5], [6]. Similarly, when a failure occurs on the selected path, several works addressed the failure recovery in SDN [7], [8], [9]. Other studies survey the existing failure recovery management. Fonseca et al. [10] presented an overview of fault management threats focusing on each layer in SDN and a threat by each interface between the layers. They further discussed trade-offs between approaches and their stabilities for different SDN applications. To ensure SDN reliability, Yu et al. [2] also presented a systematic survey of SDN fault management by evaluating the existing works, and their weaknesses were noted. In-depth analyses focusing on detecting, localizing, correcting, and preventing faults were among the issues noted and discussed extensively. Issues of fault monitoring, diagnosis, recovery, and repair were also touched up.

On the other hand, Rehman et al. [11] focus on SDN fault tolerance and classify it into four phases: Error Detection, damage confinement and assessment, error recovery, and fault treatment and service continuation. These phases were extensively discussed. Error detection highlight how failure is detected. The error recovery phase explained how to restore the system and maintain a failure-free network. The extent of damage caused by faulty components is presented in damage confinement and assessment. Fault treatment and service continuation discussed how faults are either repaired or the system is reconfigured to avert further fault damage. Finally, additional research gaps were identified. Other works by Ali et al. [12] review schemes for link failure recovery; their work investigates open research questions posed by centralized network architecture. Restoration and protection recovery were analyzed. Performance gains for the former and latter are highlighted, and research directions are presented. In a similar effort, Malik also surveys fault tolerance

related [3] and emphasizes the need to investigate restoration mechanisms.

However, research is better with time, and SDN fault tolerance issues are still in their infancy [11]. There is still a wide range of research opportunities with unanswered questions by the research community. This survey investigates routing and failure recovery issues. Failure recovery is greatly affected by several design factors: computing a reliable primary path while meeting different flow demands with efficient resource utilization, the controller's operation mode, and the failure path's recovery range. Overlooking these factors may lead to poor network resource utilization, high communication overhead, and poor network performance. Moreover, after the occurrence of network failure, Quality of Service (QoS), load balancing, and post-recovery congestion are other crucial issues that may affect the network performance in a large-scale network. Dynamic load balancing and multipath forwarding to solve the problem of congestion control are also discussed in this survey.

Transmission medium requires high throughput, resilience, and reliability for efficient data transmission. Multimedia applications such as live video streaming, video gaming, and other delay and throughput-sensitive flows require an optimal path for routing. To cope with these multimedia services demands, optimal dynamic routing is usually computed to achieve better network resource utilization, leading to higher network throughput and better QoS. Unfortunately, communication networks are prone to failure, and the selected path may not always guarantee service availability. Furthermore, the link failure between the chosen route may negatively influence network performance because it may interfere with some of its tasks, such as routing. Therefore, a dynamic routing decision is required to incorporate link quality to minimize link failure occurrence. Unfortunately, there is a lack of sufficient literature to include routing problems with failure recovery in SDN for efficient resource utilization and better QoS. Table 1 summarizes the scope of the related existing survey. This further motivates us to investigate the SDN's dynamic routing schemes and failure recovery approaches. This paper surveys routing problems and failure recovery management for efficient resource utilization.

The rest of the paper is organized as follows: Section II presents an overview of Software Defined Network and discusses its widely used standard protocols. Section III explained the research challenges and problem background related to routing and failure recovery. In addition, routing problems and related works are also discussed. Section IV surveyed failure detection and recovery approaches. Future research directions are discussed in Section V. Finally, Section VI concludes the paper.

II. OVERVIEW OF SOFTWARE DEFINED NETWORKS

The SDN is a composition of different planes, such as the Application Plane (AP), the Control Plane (CP), and the Data Plane (DP); other layers are NBI and SBI, as shown in Figure 1. Each of the planes and layers has its specific functions. AP is a set of applications such as routing policy, Quality of Service (QoS), access control list, load balancing, and others. These applications leverage the NBI to implement network control and operational logic at the CP. The CP mediates between the AP and the DP, thereby managing the entire network through programmability features using the plug-in. This way, it generates a network control function based on defined policies by the network operator from AP into a set of instructions inform of flow entries into DP to guide the network management. However, a single CP can manage significant traffic in a small network setting. Distributed controllers can also manage the network when a single controller fails. Details about distributed controllers can be found in [1]. The DP is a set of networking equipment, such as switches and routers, similarly used in the conventional network. The main difference is that DP is now simply forwarding entities without any embedded control logic to take an independent decision. The network intelligence has been removed while focusing only on delivering traffic flows. Therefore, for any control function DP consults the CP for further action. In return, it generates a set of flow entries and instructs DP to be installed in its Flowtable through SBI.

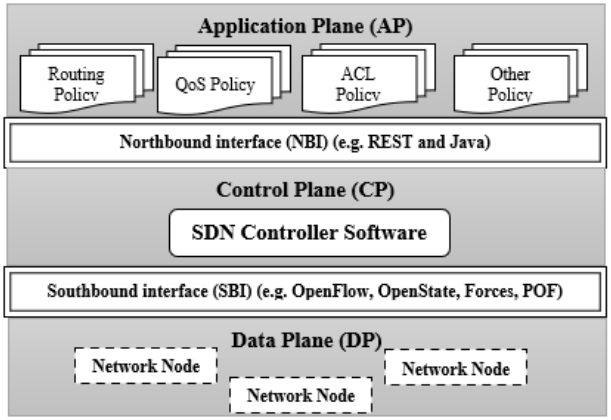

FIGURE 1. SDN architecture.

The SBI served as a critical component for separating CP from DP through programmability. Typically, the manufacturing of a new hardware switch can take an average of two years to be ready for commercialization, with upgrade cycles of up to nine months [13]. Developing a new software product may take six months to one year [14]. Therefore, this process takes a lot of time with high investment risk, which may slow network innovations. As such, the SDN SBI API emerged to offer programmable to provide flexible network management. Forwarding and Control Elements (ForCES), Open v Switch Database (OVSDB), and Protocol-Oblivious Forwarding (POF) were among the early SBI API proposals [1]. ForCES provides flexibility compared to conventional network management without changing the network architecture. However, the CP is potentially maintained in the same network element.

In contrast, OVSDB provides more advanced management features for Open v Switches. POF is among the SBI APIs aimed at enhancing the forwarding performance of DP. However, these SBI APIs rely on modifying DP to support Flowtables [1]. This way, the remote entity can flexibly configure the DP through operations such as adding, removing or modifying flow entries in the Flowtable at DP. OpenFlow is another SBI API that emerged to facilitate network and control management without altering the SDN architecture. The SDN controller has direct access and control of the DP through a logical data structure called Flowtable. The flexibility of OpenFlow attracted not only research communities but including an industry [13]. As a result, OpenFlow is widely considered the most SBI API to achieve the benefit of SDN. Although, OpenState [15] was proposed as an extended version of OpenFlow with more programmable features aimed at making the DP devices act independently without consulting the controller. However, OpenState has not been standardized as an acceptable SDN SBI. As such, OpenFlow implements the concept of SDN by abstracting network communications in the form of flows to be processed by the OpenFlow switches with a set of instructions, in other words, flow entries.

III. CHALLENGES AND BACKGROUND

various types of SDN Applications in large-scale networks generate different traffic flows, such as throughput, delay-sensitive, and bandwidth-sensitive flows. Some flows are generated by network protocols such as (ARP and DNS), and interactive applications usually generate latency-sensitive flows. Conversely, scientific computing, MapReduce, and machine migration applications formed throughput-sensitive flows. The recent proliferation of the Internet of Things (IoT) devices required network architecture to react in real-time and be scalable for many traffic flows. Forwarding devices are also required to present efficient data delivery for both types of traffics flows with an acceptable delay to meet the demand of CGNs. This way, for every arrived flow, a feasible primary path is computed considering available network resources. The shortest Path Algorithm (SPA) is widely used to select feasible paths [16] to achieve efficient network resource usage. The most common way for a SPA to achieve resource efficiency is to limit resource consumption and to keep the network load balanced. However, traffic flows differ with a variety of Quality of Service (QoS) requirements, and SPA may not meet the demand of some flows. As such, additional constraints must be imposed to meet the needs of different flows while achieving efficient network resource utilization. To achieve this, the controller needs to i) gain and maintain accurate global network knowledge, ii) compute the feasible path that meets the demand of given flows while efficiently utilizing the underlying network resource, and finally, iii) install the corresponding rules in the switch Flowtable to guide the routing process. Therefore, the efficiency of path selection heavily depends on how the controller gain and maintain accurate Link State Information (LSI). The existing SPA used static LSI (i.e., hop count, distance, link capacity) to compute a path [17], [18]. Others considered dynamic LSI (e.g., link utilization, available link capacity) [19], [20]. This way, dynamic routing periodically requests LSI information from underlying network devices. Static LSI significantly reduces communication overhead because the controller does not have to query the network state regularly. However, as the network increases in size, it will cause congestion because it always calculates the same path. In contrast, dynamic LSI yields better performance but at the cost of periodic network state overhead.

Conversely, reliability is another issue affecting smooth data transmission after computing the feasible path. Link or switch failure is reported to have occurred frequently, on average, every 30 min [7]. In addition, some links are likely to fail due to software bugs or hardware malfunctions. Upon failure occurrences, all traffic flows affected by the failure must be rerouted through backup paths as soon as possible for continued service provisioning. For CGNs, the failure recovery process must be completed within a predefined time interval [18]. The recovery mechanism in SDN is divided into two categories: restoration and protection. In the case of restoration, the backup path is deployed on-demand. As for protection, the backup path is always pre-planned in advance before occurrences of failures. As a result, disrupted traffic flows can immediately be redirected without incurring extra signaling overhead. Therefore, protection approaches can meet the demand for delay-sensitive application requirements of CGN. However, the switch Flowtable storage Ternary Content Addressable Memory (TCAM) is a constraint with limited space [21]. Typically, a Flowtable is populated with correlated primary and backup path rules for fast-forwarding. Unfortunately, the corresponding forwarding rules is relatively higher than the TCAM capacity.

In contrast, the restoration approach reduced the number of forwarding rules and can adapt to frequent network changes. However, the merit comes at the cost of communication overhead and sometimes longer computing paths. Finding the fastest path could be expensive in terms of the time it takes to compute a new route and modify the relevant switches Flowtable [22], [23]. The number of forwarding rules in the Flowtable can contribute to higher update operations. Switches with many forwarding rules in their Flowtable experienced higher update operation, which may extend the routing convergence time. Latency is critical for some applications such as VoIP (Voice over IP). Traffic from delay-sensitive application flows must arrive at their destination without considerable flow setup latency, typically within 25 milliseconds [24]. Therefore, path computation and failure recovery approaches using either restoration or protection still face various challenges in the static or dynamic routing and failure recovery process [25].

For the routing issues, network unbalances and congestions are among the issues that affect static routing, although they can be overcome by dynamic state routing. However, it required substantial overhead to obtain accurate

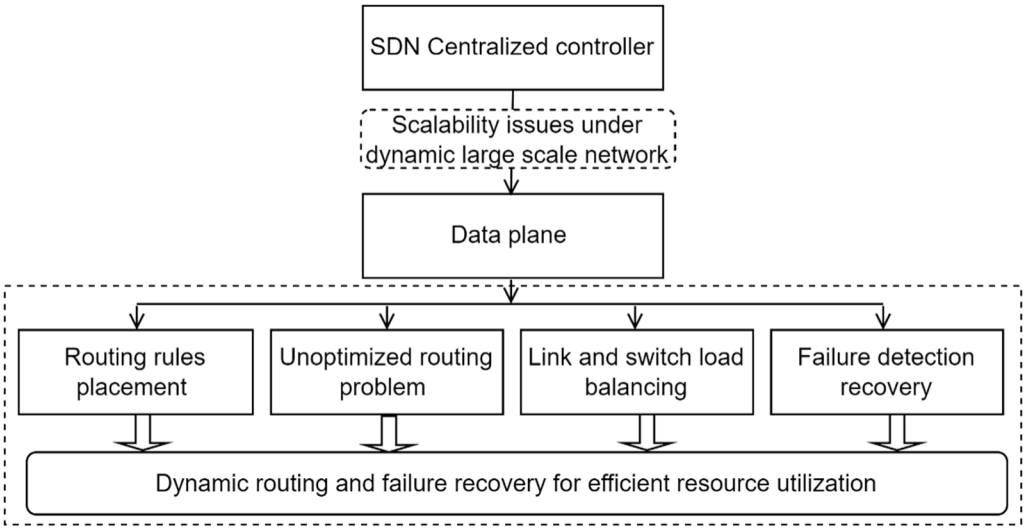

FIGURE 2. Routing and failure recovery challenges under dynamic and large-scale network conditions.

dynamic link state information. In addition, link cost needs to be regularly recomputed based on the new network state, while a prior study has shown network state changes every 1.5 sec [26]. This could lead to extra pre-computation costs before the actual path computation. In full protection coverage, network topology density differs, making full protection very hard when multiple components fail. Typically, failure can be categorized into two parts in SDN: 1) at the controller and 2) at the data plane. At the controller, master and slave controller failure problems occur when one of the controllers runs out of the assigned capacity [27]. Several controller failure problem solutions have been introduced over the years [28], [29], [30]. However, multiple controller failure problems are out of the scope of the present document. We focus on the failure at the data plane; this way, component failure can be categorized into four types: single-link failure, single-node failure, multiple-link failure, and multiple-node failure. In highly dense network topology, some flows may not always have a valid alternative path for multiple component failures; forcing the controller to reroute the affected flows may lead to routing loops. Hence, such flows may not have full protection coverage, recovery can be pretty challenging when alternative path is not provisioned efficiently.

Unfortunately, an alternative path may not always be available when needed and could fail earlier than the primary path [26]. The alternative path performance is usually affected by many factors. The most notable one reflects on bandwidth utilization, delay, and packet loss rate, and the worst case is chosen path, which correlates with the primary. Failure on such path may significantly affect many paths in dense network topology. Therefore, when selecting an alternative path, it is essential to consider the current performance of links and the service availability as the traffic evolves.

Additionally, the switch Flowtable storage constraint is another factor that affects rerouting during failure recovery convergence time. Most recovery approaches provisioned flow entries to protect each flow; however, such decisions lead to many entries, which consumed Flowtable storage space. As a result, some Flowtable entries will be removed when the storage usage crosses the threshold or is exhausted, leading to a Flowtable scalability concern, especially in large-scale networks. This survey organized these challenges into four categories, as illustrated in Fig. 2. Routing rules placement, unoptimized routing, link and switch load balancing, failure detection, and recovery.

A. ROUTING RULES PLACEMENT

SDN is flow-based routing instead of destination-based, which is widely used in traditional networks. The corresponding flow rule is installed in the switch Flowable memory for each arriving traffic flow. It is a logical data structure used in OpenFlow switches to handle flows. This data structure used special high-speed memory to called Ternary Content Memory Addressable (TCAM) on switches to ensure matching flexibility and high lookup performance in constant time [31]. However, although TCAM has a high speed of searching for flow rules, it is 400 times more expensive with limited storage space than RAM-based storage. Besides, the size of each flow rule is 356 bits, much larger than the 60-bit entries used in conventional switches [31].

Conversely, per-flow routing required many dedicated rules placement to efficiently route flows to various destinations. As a result, the data forwarding entities need a large TCAM memory size to accommodate many flow entries. To overcome this shortcoming, the SDN controller can reactively install the forwarding rules on demand upon arrival of each traffic flow. Unfortunately, in dynamic large-scale

networks where traffic flows frequently change, the number of flow setup requests will augment rapidly, increasing the processing load on the SDN controller, which causes extra packet processing delay [32]. This overhead and delay can significantly affect the real-time application and degrade network performance.

Software switches based on commodity servers have recently gained popularity. These switches have a large storage capacity and can handle packets quickly (e.g., 40 Gbps on a quad-core machine) [33]. However, software switches are more constrained in forwarding and lookup rate than commodity switches [34], because they employ general-purpose CPUs for forwarding. In contrast, commodity switches used Application-Specific Integrated Circuits (ASICs) designed for high-speed throughput. In addition, software switches stored routing rules in computer Random Access Memory (RAM) which has a larger capacity; unfortunately, the lookup speed in the Flowtable is slow. In contrast, the lookup rate in TCAM-based Flowtable is faster at the cost of a small space. Furthermore, the proliferation of IoT devices further aggravates the TCAM storage problem because of the large number of traffic flows generated by different devices.

To overcome the storage limitation, the SDN controller configures flow rules with an idle timeout with a small value [35]. It is the inactive period, after which the flow rule will be evicted from the switch Flowtable. Unfortunately, traffic flows exhibit some variables with different inter-arrival times and duration, which make the idle time value inefficient. Alternatively, the controller configures flows with a hard timeout value [36]. It is the total life span of flows after which the rule is declared invalid and removed from the Flowtable. However, a hard timeout may be too long for some flows, mainly with less or no packet to match. As such, flows will stay in the Flowtable longer than necessary, occupying precious space. Therefore, the diversity and heterogeneity of IoT devices and the traffic flows generated with different variabilities can easily overwhelm the timeout-based flow-rules placement mechanism [1]. Various schemes have been proposed to address these challenges. For example, the work in [37] proposed a timeout scheme based on flow inter-arrival time and duration to manage the switch Flowtable efficiently. Another proposal explores the impact of timeout settings on Flowtable [38]. Therefore, we extensively survey the various schemes proposed in the literature to address the Flowtable table memory limitation in the context of the routing rules placement problem.

B. UNOPTIMIZED ROUTING PROBLEM

SDN controllers make routing decisions on behalf of the data-forwarding entities using a set of policies. These policies are converted to instructions and sent to forwarding elements such as switches and routers. The endpoint policy determines each flow's source and destination nodes based on the supplied high-level design requirements, while the routing policy determines the flow path. This way, the shortest-path routing policy asks the network forwarding element to forward packets along the shortest path between two given nodes. To efficiently route flows, the SDN controller must regularly monitor and gain accurate knowledge of Network State Information (NSI). Such knowledge is used to compute the best path to meet the requirement of given flows while efficiently utilizing the network resource. Afterwards, place the routing rules on the switches Flowtable to guide the routing process. The ability to compute a feasible path heavily depends on how fast the controller obtains the NSI. To obtain the NSI, the existing routing algorithms used either static or dynamic link state information [39]. The former calculates the link state information once during the topology discovery state and computes a feasible path based on hop count, distance, and link capacity. While the latter considered available link capacity, link utilization, or the number of processed flows. This way, the controller periodically queries the underlying network device to obtain updated network information. Static routing has less overhead because it does not always request updated information from switches. However, it always finds the same path for all flows, making it infeasible for some flows with special QoS demands. Besides, it causes congestion in some links while others will be underutilized. As a result, researchers improved the routing algorithm using dynamic routing. Although dynamic routing may adapt to real-time traffic having different QoS demands, there is significant overhead in frequently querying switches to obtain updated network information. Besides, it is very challenging to get an accurate NSI because of the delay and message exchange overhead. An inaccurate NSI may lead to accepting more flows than the network resource can handle, and significant packet loss is unavoidable.

Over the past years, several routing schemes have been introduced to improve dynamic routings while efficiently utilizing network resources. Unicast and multicast are widely used as the two classes of routing. A unicast routing problem is defined as given source s to destination t, with a set of best effort/ QoS constraint C, and possibly an optimization goal to find the feasible path from s to t while optimizing C. Multicast routing problem, given a set of R destinations and set of X constraint such that source s, and all set of R destination satisfy X constraint. This way, residual bandwidth is considered one of the main problems for QoS route computation. We classified the existing routing problems into three categories, as shown in Fig. 3.

Other researchers leverage machine and deep learning models to devise routing schemes [40]. Several proposals were introduced to incorporate a deep learning model on SDN [41], [42], [43], [44], [45]. Unfortunately, this paper does not cover machine and deep learning approaches.

1) ROUTING WITH LINK COST AWARE

The minimum Hop Count Algorithm (MHCA) [46] selects the path with the shortest distance between the given source and the destination. MHCA maintains information about each link; only those with enough resources to meet the user's needs are considered for routing. MCHA is a straightforward

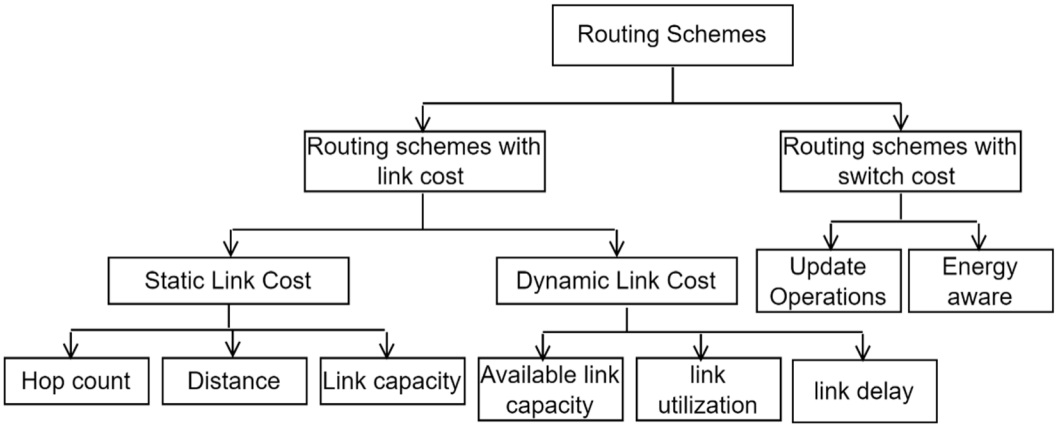

FIGURE 3. Routing schemes taxonomy.

algorithm and can easily be implemented. However, it might easily lead to a bottleneck for upcoming requests due to inefficient use of network resources. Widest Shortest Path Algorithm (WSPA) was presented in [47]; it is a modified version of MHCA which chooses the path with the highest amount of bandwidth from the set of available paths. MHCA exhibits a trade-off between load balancing and resource consumption. Shortest Widest Path Algorithm (SWPA) [48] is another variant of MHCA, it computes the widest possible path among N paths, and the path with the smallest number of hops is considered optimal. However, most static routing assumes the controller has full knowledge of the network topology and overlooks the dynamic link state information. The controller can calculate the shortest path based on distance or hop count using the discovered topology information at time t_1. Unfortunately, such an assumption may not yield optimal performance, especially in dynamic large-scale networks. Dynamic link state information routing was devised to overcome the challenges of static routing. In this way, several schemes were introduced to optimize the Quality of Service (QoS) parameters [39], [49], [50], [51]. State-of-the-art dynamic routing focused on minimizing the interference among the flows [16]. The authors developed a heuristic path selection algorithm to avoid routing flows through critical links. It is heavily loaded links that consume a significant amount of bandwidth, and routing flows through a path with such features would make it quite challenging to satisfy the future demand of certain flows. Although the scheme outperforms minimum hop and widest routing algorithms. However, traffic flows are often unevenly distributed, bypassing critical links at time t_1, and routing flows through another path may not always guarantee the network load balance. The selected path may be overloaded at time t_2 and, consequently, be problematic to multimedia applications that require certain QoS demand. As such, it is desirable to incorporate more routing metrics to maximize network resource utilization since it is the primary concern of the infrastructure provider. The works in [19], [52], and [52] examined the effectiveness of various routing algorithms for dynamically establishing performance-guaranteed traffic while considering bandwidth and path latency besides bandwidth rejection ratio. The scheme proved that bandwidth-constrained algorithms using source-to-destination pair information could considerably improve network performance. However, it chooses long routes regardless of network load, which could affect delay-sensitive applications. Their extended work in [53] classified flows based on their level of delay sensitivity aimed at optimizing resource utilization while providing absolute bandwidth and delay guarantees. However, multicast applications require high bandwidth and lower delay while maintaining low controller computational overhead. Unfortunately, their solution exhibits high complexity as the number of delay-sensitive application flows increases and consequently declines the system throughput. If the network fails to distribute bandwidth for flows correctly, the performance of delivering and processing data will be degraded due to excessive network congestion. To fulfil the bandwidth and delay requirements of multicast applications and apply QoS parameters based on the current availability of network resources. The work in [51] and [54] presents a uni-cast and multicast request to increase network throughput under critical and user-requested bandwidth. An application-aware routing was presented in [55], [56], and [57]. The schemes compute the link load at a regular time interval. After the arrival of flows, they examine the flow's connection type and bandwidth requirement. This way, they allocate paths accordingly. Dynamic and adaptive multi-path [20] is another application-based routing that computes paths based on packet loss, time delay, and bandwidth for multimedia applications. In contrast, the work in [58] argues that the composite routing metrics may not yield better forwarding performance, the greater the probability that the device will send flows over the most optimal path. This way, they devised an adaptive routing scheme considering delay, packet loss, and jitter.

The model adaptively routes flows based on their QoS demand. Although there was forwarding performance gain, calculating the metric values may induce another overhead as the network increases in size. Other solutions [59] compute paths based on link load to avoid congestion. A critical switch and link routing scheme was presented in [60] and [61] to compute the path based on link and switch features. This way, flows are routed through a path with minimum critical switches and links to improve the packet delivery ratio and throughput.

Although, the aforementioned literature has significantly made a difference compared to static routing. However, Inaccurate network information and protocol overhead are among the pressing issues that affect dynamic routing [62], [63]. The references [19] and [64] evaluate static and dynamic routing under two scenarios 1) where the accurate network state information is available as assumed in the literature and 2) considering the practical case where the controller periodically gathers the network state information with inaccuracy. Although the authors noted the former is impossible in reality. However, dynamic link routing outperforms static in the number of accepted flows and total throughput. In addition, they observed that the performance of every algorithm is adversely affected by the inaccuracy of network state information. Other researchers argued that flows were overrated by only considering their QoS requirement. The common approach of routing flows based on bandwidth, delay, throughput, loss, and other routing metrics, may not always be realistic. However, flows on the internet exhibit variabilities; some network applications generate flows with a large number of packets others contain a small number of packets. Flows are typically classified into two types: elephant flows and mice flows. The former are not many, but they have high traffic quantities, contributing significantly by increasing link utilization. While the latter are large, they dominate the switch Flowtable entries.

Therefore, managing the influence of both flows could go a long way in balancing Flowtable or link resource utilization. Hedera [65] introduced a dynamic flow scheduling approach to minimize conflict between elephant and mice flows while improving resource utilization. An OpenFlow-based architecture was devised to dynamically modify flows according to their traffic load. Similarly, MiceDCER was introduced in [66] to promote mice flows by assigning internal Pseudo-MAC (PMAC) addresses to the edge switches and hosts. Mice flows are typically associated with latency-sensitive and bursty applications like VoIP and search results. They contained a small number of packets but many flows live for a short time. The authors leverage wildcard features to aggregate flows to reduce the number of corresponding forwarding rules. This way, the number of rules was optimized with high throughput.

Similarly, DIFFERENCE was presented in [67] to dynamically sets up paths for elephant and mice flows separately. DIFFERENCE estimates residual link utilization at regular intervals. This way, mice flows are routed based proactively approach, while elephant flows are forwarded through the least congested path. Although, path search space was improved while guaranteeing bandwidth requirements. However, aggregated routing metrics were not considered for optimal network performance. In addition, switch Flowtable resource and link utilization were overlooked. Imbalances of these resources could potentially harm network performance [68]. A DIFF was introduced [69] to differentiate flows based on their impact on a network resource aimed at balancing switches, Flowtable occupancy and link utilization. This way, DIFF adaptively selects routes for elephant flows based on current link utilization in the network to achieve high network throughput.

However, these approaches are Dijkstra-based based routing solutions. The complexity of the Dijkstra algorithm is that the number of nodes and edges in the network affects the algorithm's efficiency [70]. Alternatively, Researchers apply meta-heuristic techniques within the routing optimization algorithms in SDNs to solve this complexity. Ant Colony Optimization (ACO) is the most well-known meta-heuristic and is widely used for routing optimization. Interestingly, its performance was tested and outperformed other traditional routing techniques. ACO methodologies also support flow-based routing strategies as used in SDNs. The references [71], [72], and [73] were introduced to manage an elephant and mouse flow based on ACO. Link delay and bandwidth are employed as reference indicators of the transmission path. Mice and elephant flows are routed through the best path that meets their flow demand.

Similarly, the work in [73] presents QoE-centric flow routing. Their work focus on identifying the best available routes for various multimedia services. QoE depends on QoS, and link resource constraints such as delay, jitter, and packet loss were incorporated in path computation for audio, video, and file transfer. Each flow was routed according to their requirements while considering the network limitation. The works in [71], [72], and [73] may perform well in a small network. However, due to time and space complexity, they may not guarantee the same performance in a dynamic large-scale network.

In contrast, Hybrid Ant Colony Optimization (HACO) algorithm [50] was proposed to address the issue of time and space complexity. A box-covering and k-means clustering methods were used to divide the network into the small subnet. As a result, flows are routed to the best path while optimizing the computation time. Although, there was performance gain in terms of loss ratio and delay. However, their solution and [71], [72], and [73] may impose some extra processing load on the controller because the network changes more often. For every subtle change, the heuristic algorithm has to be triggered, which in turn may not only affect the performance of the SDN controller but will also throttle the switch update operations.

Therefore, different approaches were proposed to improve routing efficiency in SDN, as summarized in Table 2; some schemes generalized flows to achieve QoS requirements.

TABLE 2. Comparison of routing schemes.

Related work	Routing Information		Routing Metrics				
	Lins state information	Aggregate metrics	Path Delay		Bandwidth	Packet loss	Others
			Link delay	Switch update			
Millard *et al.* [46]	Static	X	X	X	X	X	distance
Zheng Wang [48]	Static	X	X	X	X	X	hops
Mulai *et al.* [16].[39]	Dynamic	X	X	X	√	X	X
Tumovic S. *et al.* [52][53]	Dynamic	X	√	X	√	X	Rejection ratio
M. Huang *et al.* [51], [54]	Dynamic	X	√	X	X	X	Throughput
M. Beshley et al[58]	Dynamic		√	X	√	√	Jitter
Mohammed *et al.* [65]	Dynamic	X	X	X	√	√	x
E. Akin *et al.* [19] [64]	Static and Dynamic	X	X	X	√	√	Throughput
F. Su ´ arez *et al.*, [66]	Dynamic	X	√	X	√	X	X
H. Zhang *et al.* [67]	Dynamic	X	x	X	√	X	X
L.Cheng and .Wang [55]	Dynamic	X	X	X	√	X	Throughput
W. Jiawei *et al.* [20]	Dynamic	X	√	X	√	√	X
R. Ramirez *et al.* [55] [56]	Dynamic	X	x	X	√	X	X
C. Wang *et al.* [71] [72], [73]	Dynamic	X	√	x	√	X	X
O. Dobrijevic et al. [73]	Dynamic	X	√	X	X	√	Jitter
N. El-Hefnawy et al. [50]	Dynamic	X	√	X	√	√	x

Other literature leverage a heuristic algorithm to improve time and space complexity. However, these approaches do not relate to switch resource constraints like switch updating time while devising their schemes. It is an important parameter that directly impacts the routing algorithm because corresponding forwarding rules need to be placed on the switch routing table to guide the routing process. Therefore, overlooking such metrics may directly affect the routing convergence time. In the following, we have discussed other related works that considered the switch resource.

2) ROUTING WITH SWITCH COST AWARE

Routing policies required corresponding forwarding rules installed in the SDN switch memory to route flows efficiently. In contrast, the number of routing rules in dynamic large-scale networks keeps increasing. It leads to poor resource utilization, which needs urgent attention to meet the demand of users. Several solutions have been proposed to improve this precious resource utilization over the years. In the following sections, we divide the existing literature into two categories Routing with Flow operations and Energy awareness to enhance the switch memory utilization. In addition, we summarized the most related work in Table 3.

A. ROUTING WITH FLOWS OPERATIONS AWARE

Since SDN is flow driven network, forwarding devices regularly carry out update operations in their routing table to guide the routing process. Several approaches were introduced over the years to incorporate the switch cost during routing decisions. In a dynamic large-scale network, thousands of flows could easily be disrupted, such flows must be restored within the shortest possible time, and the time required to perform such operation is significant and, therefore, must be optimized to maintain a carrier-grade network status. STAR [74] is an online routing scheme to efficiently utilizes limited Flowtable resources while maximizing network performance. STAR frequently detects switches' real-time utilization of Flowtable and intelligently removes inactive forwarding rules to accommodate more new flows. This way, routing paths for new flows are computed based on switches real-time Flowtable usage. The references in [75] and [76] formulate a problem to find an optimal path with the lowest path cost and update operation. They compute N number of paths and compare it with their operation cost. The path with fewer operations and a threshold value is chosen to forward flows. Although the selected path may speed up routing convergence time. There is a lack of information on how they arrived at operational cost estimation. In addition, computing threshold values more often may not guarantee the feasibility of an end-to-end path in a large-scale network [22]. The work in [23] finds a reliable path based on a disjoint path aimed at reducing the number of update operations. They select the path with a set of nodes frequently shared by many paths. This way, the number of forwarding will be reduced. However, such a design may not guarantee the reliability of the link as the network evolves, some switches could easily be overloaded, leading to high congestion. RAF [77] calculates link reliability and installs minimum flow rules for multiple paths based on the reliability value of the path. Such value is appended to N paths, and the path with a higher value is chosen. However, this approach may involve higher reliability value computation costs in large-scale networks.

In contrast, the approach in [22] noted switch update operation is proportional to the length of the selected path; the longer the path, the higher the operation cost. They devised an approach based on the graph theory for E2E path computation

TABLE 3. Comparison of routing with switch cost aware related work.

Related work	Aggregated routing information	Routing metric cost		Weakness
		Switch update cost	Switch Energy-aware	
S. Astaneh *et al.* [75] [76]	X	√	X	Computing threshold value more often more not guarantee the feasibility of an end-to-end path
A.Malik *et al.*, [23]	X	√	X	Node on a disjoint path can lead to congestion
S. Raza et al. [77]	X	√	X	Reliability computation value imposed extra processing load on the controller
A.Malik *et al.*[22]	X	√	X	The solution does not secure the shortest path from source to destination
L.Luo *et al.* [80]	X	√	X	This strategy is constraint by the bandwidth limit function
H. Xu. *et al.* [81] [82]	X	√	X	The solution may require a large solver to converge in a large-scale network.
Z. Zhao *et al.* [84]	X	√	X	an ILP method is not scalable for large dynamic network-scale
R.Biswas and J. Wu [79]	X	√	X	The controller may not regularly have flows visibility
S. Kotachi *et al.* [78]	X	√	X	sharing a single entry for many flows may introduce congestion
Guo *et al* [74]	X	√	X	It focuses on routing based on Flowtable utilization

to divide the network into communities. Suppose any part of the network is disrupted due to link failure. In that case, only the forwarding rules of the affected community will be updated, and the rest of the forwarding rules will remain intact. This has minimized the flow operations cost. However, their solution does not secure the shortest path from source to destination, and the process of detecting the affected community and the failed link may incur an extra processing delay. A multicast routing model for multiple multicast requests to reduce the number of rules was presented in [78]. The authors formulate an ILP model to concurrently determine several multicast tree paths to share a single flow entry stored in a forwarding element set up for a single receiver. Although, the approach can reduce the number of operations. However, sharing a single entry for many flows may introduce congestion, and changes in flow behaviours may affect many paths. To address this challenge, reference in [79] formulates a problem to minimize the number of rules to redirect flows. The former examines link capacity, and congested links are bypassed, while the latter group multiple flows and merges their forwarding rules. This way, they routed flows without incurring much flow operation. There was performance gain for these solutions in terms of reducing the number of flows operation. However, it is quite challenging for SDN controllers to regularly have global network knowledge with flow aggregation techniques. Because flows are treated using wildcard rules, its additional features in OpenFlow which may not be supported by some switches [13], [31].

Therefore, most current solutions take a long time to minimize the update operation or introduce controller overhead, making them impractical for large-scale high-dynamic networks. To overcome these challenges, FLUS was presented in [80]; a Segment Routing (SR) based strategy was introduced for fast and lightweight path update operations. For any change in the network, FLUS immediately uses SR to develop a new path by joining some parts of existing and new paths. Afterwards, the actual path for the data transmission will take place. This way, flows are shifted to the newly computed path. Unfortunately, this strategy is constrained by the bandwidth limit function. The references in [81] and [82] proposed an ILP model for a real-time delay-optimized flow route algorithm. The approach separates the forwarding strategies of various flows based on their sensitivity to delay. The authors claimed to reduce route update operation by 60% compared to benchmarking work. However, the solution may require a large solver to converge in a large-scale network.

Other solutions [83], [84], and [85] focus on aggregating the number of flows with similar features going to the same destination to reduce the number of flows operation. For example, an Integer Linear Programming model (ILP) was introduced in [84] to minimize the total cost. First, it assigned predefined weights to differentiate, thereafter, aggregate flows with similar features. Afterwards, they dynamically route flows to their destination. However, an ILP solution is not scalable for dynamic large-scale networks.

B. ROUTING WITH SWITCH ENERGY SAVING AWARE

It is worth noting that routing rules are stored in the switch memory, usually implemented in TCAM. It is good in terms of look-up rate and forwarding performance. However, it's known for its high cost and power-hungry. Therefore, the proliferation of traffic flow load on communication links influences the energy consumption of links. Similarly, the hardware constraint of TCAM further argument the power consumption of forwarding elements in SDN. To overcome these challenges, various Energy Aware Routing (EAR) was introduced over the years to minimize the energy consumption of these resources while preserving connectivity and QoS, as summarized in Table 4. Energy Aware Routing with Compression (EARC) was proposed in [86] and [87]. It's an ILP that uses a greedy heuristic to optimize switch resource energy consumption using a software switch.

High cost was eliminated with faster Flowtable update operation up to 10 times faster than harder ware switches. However, the scheme is associated with packet processing delay.

MINNIE [83] was introduced to reduce the extra delays for routing lookup rules while reducing TCAM energy. It provides a two-phase solution: the compression and the routing phase. The former uses a heuristic method to compress forwarding rules with the same feature, while the latter provides a heuristic based on the shortest path algorithm with adaptive metrics. This way, it reduced the number of routing rules by 50%. However, MINNIE will experience the same weakness as the works in [86] and [87] due to their architectural design of hardware and software switches. The processing of exchanging packets between the switches will no doubt introduce packet processing delay. Reducing routing rules reconfiguration cost is another way to optimize switch TCAM power consumption. Reference [88] presented an LP optimization model to reduce the cost of re-configuring flow tables when traffic demand changes. An LP model was used to formulate the problem while considering the obsolete flow entries that must be removed and the new flow rules that must be installed. Furthermore, a Genetic Algorithm has been presented for decreasing network power usage while minimizing the number of updated forwarding rules in a Flowtable.

Since switch TCAM power consumption is correlated with the flow's arrival, other solutions focus on devising traffic-aware energy saving. The key idea is to turn on or off network components (for instance, some switches) based on the traffic load. For example, when traffic loads are low (especially at night), this strategy can save up to 50% of overall energy utilization, like the work in [89]. Typically, an elastic structure is used to depict network components that can expand and contract in response to dynamic traffic loads. The main problem is deciding which components to switch on and which one to turn off without compromising the desired level of QoS [90]. Elasticity, topology awareness, queue engineering, and smart sleep on and off are desirable qualities of a traffic-aware controller. The ability to dynamically increase or decrease the number of network components employed in response to traffic is referred to as elasticity.

Topology awareness adds the advantage of using formulations and solvers to customize to any given topology. The hierarchically arranged fat-tree architecture is the most commonly employed in data centres. Knowing how the components are grouped and their capacities allow us to take alternative path that avoid energy-intensive paths. The work in [91] presents an EAR by devising four modules on the SDN controller using fat-tree topology: optimizer, routing, flow monitoring, and power control. It periodically receives input information from flow monitoring and finds the most energy-efficient subnet that satisfies current traffic demand. Afterwards, the optimizer provides the active topology to the routing and power control module. Thereafter, the power control module changes the power state of the switches and link cards while the routing selects the optimal path for flows. This strategy improved the power-saving level with efficient network resource usage.

Queue Engineering approaches provide extra port-level traffic monitoring functionality. Flow size and link bandwidth could be obtained at regular time intervals. This way, the ending time of flow can be deterministically calculated to schedule unallocated flows in queues to increase the flexibility of the flow scheduling method. The references in [92] and [93] presented a routing strategy that combines exclusive routing and flow scheduling to achieve efficient energy saving. The scheme expands flow scheduling to the time dimension and considers energy consumption during flow transfer time. The active and suspended flow sets are used to schedule and transfer flows as part of a heuristic search for the flow group solutions that use the least energy. Another solution in [94] reduced network power by taking Flowtable size and link bandwidth into account for a single flow per user. This study employs Dijkstra's method to minimise network power upon flow arrival while considering link bandwidth and Flowtable size. They employs ACO to determine the optimum path and minimize network power for all flows. However, the flow arrival in a dynamic large-scale network is beyond single flows. Hence, such a solution may not be applicable in different network settings.

Moreover, link bandwidth and Flowtable size are not adequately considered while employing the ACO. To overcome this limitation, reference [95] modified and considered link bandwidth and Flowtable size while routing every flow. Instead of relying on ACO, it selects several routes per flow and computes network power usage while considering bandwidth and Flowtable size. However, the performance gain was observed compared to [94]. However, the routing decisions are subjected to bandwidth and delay only. The dynamic large-scale network contains different applications with various flow QoS requirements. Diverse routing metrics make it more flexible to meet the demand of other flows. The Minimum Criticality Routing Algorithm (MCRA) is proposed [96], along with an energy-efficient optimization. MCRA determines the available paths based on the end-to-end request. Suppose the discovered paths do not satisfy specific requirements, such as the maximum latency and link utilization ratio. In that case, the rerouting process is initiated using the Energy-Efficient Multi-constraint ReRouting (E2MR2) protocol. However, network changes more often, as such the model needs to run each time when there is network changes. This may introduce unnecessary delay, especially for delay-sensitive applications.

However, topology-aware and queue engineering approaches are most tailored toward a specific topology, such as a fat tree. Unfortunately, such solutions cannot be applied to other topological structures. While individual heuristics work well for a specific case and cannot be generalized, broad heuristics do not fully grasp energy capacities [97]. Other solutions focus on a smart sleep approach to improve energy-aware routing. Smart sleep and off refers to the capacity to

TABLE 4. Comparison of energy-aware routing-related works.

Related Work	Method	Routing Awareness	Resource (Switch /link)	Weakness
F. Giroire *et al.* [86] [87]	ILP rule optimization model	Rules	Switch (forwarding rules)	Packet processing delay
J. Galán-Jiménez *et al.* [88]	LP model	Rules	Switch rule reconfiguration cost	LP solution required a faster solver to converge in dynamic large-scale networks
M. Rafai *et al.* [83]	MINNIE rules compression method	Rules	Switches	Two switches architectural design introduced extra flows processing delay
H. Zhu *et al.* [89]	Joint flow routing	Traffic Flows aware	Switch and link	deciding which components to switch on and which to turn off without compromising the desired level of service (QoS)
T. Manh Nam *et al.* [91]	Fat-tree architecture	Topology aware	Switches resource	Periodic flow monitoring may not always give accurate topological data.
G. Xu. *et al.*, [92][93]	Switch port information	Bandwidth-aware EAR	Link resource	The approach is a constraint to fat-tree topology which may not yield better performance in other topological structures
D.Jiang et al. [96]	Energy-Efficient Multi-constraint ReRouting (E2MR2)	Energy (ALR) Load Balancing	Link latency and utilization ratio	An optimization model needs to run each time the network changes
R. Maaloul *et al.* [99]	Integer linear programming model	EAL	Switch /link	The method is less promising for real-world large networks with traffic patterns variabilities as a result of high temporal complexity
A. Amokrane *et al.* [94]	ILP, ACO	Energy Saving (ALR)	Link Bandwidth and delay	The ACO model did not properly consider the Flowtable size and link bandwidth
M.N Siraj *et al* [95]	Energy Aware Dynamic Routing (EADR)	Topology aware	Link Bandwidth and delay	Inaccurate dynamic network state information may lead to a false negative.
B. Ozbek *et al.* [98]	Shortest Path, Genetic Algorithm (GA)	Energy Saving (sleep mode)	Link bandwidth	limiting QoS routing metric decisions to only bandwidth may not be a valid assumption.
S. Torkzadeh *et al.* [100]	A graph-based Ant Colony Optimization	Energy Saving (sleep mode)	Switches	May require a longer time to converge in dynamic large-scale networks.

turn on/off switch ports, links, or the complete switch in response to traffic. In [98] a routing method based on the shortest path is proposed for incoming flows to reduce the number of active OpenFlow switches in an SDN. This technique focus on meeting the demand of throughput-sensitive flows while considering the implicit link capacity limits. This way, they analyzed the network's night-time traffic, and the number of flows is considered negligible. This technique assumes no QoS constraints other than bandwidth for path selection. Even though energy usage was improved, limiting QoS routing metric decisions to only bandwidth may not be a valid assumption.

In [99] an energy-aware routing algorithm for SDN-based carrier Ethernet networks was proposed. It presents a binary linear programming approach for the EAR problem that optimizes the number of network switches that can be turned off while considering traffic demand and rule space limits. Unfortunately, this method is time-consuming due to the use of the Integer Linear Programming (ILP) model, a set of first-fit heuristic algorithms to reduce computation time is also offered. Interestingly, it balances energy savings and connection utilization while ignoring the application flows' QoS requirements. However, the method is less promising for large real-world networks with traffic patterns variabilities due to high temporal complexity. The work in [100] presents a two-phase SDN-based routing technique that minimises energy usage while maintaining a given degree of QoS for user flows. A minimal graph-based Ant Colony Optimization (ACO) technique reduces network energy usage. It prunes and optimizes the network tree by shutting off unneeded switches and providing an energy-efficient subgraph in charge of routing the current flows.

3) LOAD BALANCING SCHEMES

The heterogeneity of the current network devices and the number of internet-connected users increases the traffic flows generation. Since SDN is a flow-driven network, many routing rules will be placed on the set of switches to guide the routing process. However, path roles differ, some paths are frequently used others are underutilized. As a result, forwarding entities on the frequently used path may be overloaded with many routing rules, while others may have a minimum and therefore cause a load in balance. Similarly, the links connecting these sets of forwarding entities are constrained with limited bandwidth, while bandwidth demand is increasing due to the surge of traffic demand. As a result, frequently used links consume a significant amount of bandwidth. Therefore, effective load balancing is required to improve network resource usage. The load balancing technique manages incoming traffic flows, thereby distributing and sharing the traffic load fairly among the network forwarding element to improve network service availability. This way, traffic bottlenecks on data forwarding entities and communication links could be avoided. Several efforts have been made

lately to balance the network on either link load or switches Flowtable [101], [102]. For example, the approach [103] devised an Automatic Re-routing with Loss Detection architecture. Packet loss is detected with the queue stat message of OpenFlow protocol, and then the re-routing module tries to find a bypass path and applies it to Flowtables to balance the load on the switches better. However, the approach may require significant bandwidth resources in a network with varying traffic patterns.

Moreover, flow rerouting without considering the current network statistics, may results in congestion propagation [104]. In contrast, the work in [105] proposed a scalable congestion control protocol to reduce Flowtable overflow, thereby reducing queuing delay under busty traffic. The number of TCP flows passing through each switch port was monitored to ensure total bandwidth utilization does not exceed the bandwidth-delay. This information was eventually passed to each TCP source by updating the advertisement window field in the TCP header. The proposed algorithm transfers the minimum number of flows from the congested link to the backup path, resulting in improved QoS and congestion control. To alleviate the congestion and improve application performance. Multi-path routing algorithms can distribute traffic over diverse paths optimally than simple solutions like ECMP. This way, Kanagavelu et al. [106] proposed a local rerouting mechanism in SDN-based Data Centre networks to effectively manage congestion in the event of link congestion or failure. SDN controller periodically gathers port and flow statistics from all switches at a fixed time interval. Afterwards, the routing engine computes a less loaded path between pairs of demand. Link congestion was checked periodically based on a threshold value; once link congestion exceeded the T value, the SDN controller re-routed the affected flows along an alternative path.

Similarly, the work in [107] proposes an effective routing mechanism for link congestion avoidance in SDN. The controller observes the current traffic of switches and updates the topology according to the weight assigned based on computed bandwidth usage. Traffic from congested links is instantly redirected to the available links to enhance the efficiency of link usage. Interestingly, the method effectively allocates and utilizes the network bandwidth but at the cost of monitoring overhead. Another solution in [108] leverages the flow parameter to devise an algorithm for avoiding congestion. The network state is monitored by calculating the link utilization periodically and redirecting the newly arrived flows to a congestion-free path. Congested paths are obtained through threshold values. If the average link utilization exceeds a threshold, the controller predicts the congestion on the link and calculates the load to be redirected to other backup paths. This way, flows are rerouted to the proper backup path without much congestion. However, Sminesh et al. [104] argue that congestion is far beyond balancing the network on either link or path; therefore, most load-balancing schemes overlook addressing congestion propagation. The authors proposed a method that categorized links for efficient load balancing. Toward this goal, the scheme monitors the utilization of each link, and over-utilized links that cause network congestion and packet loss are identified as bottleneck links. Afterwards, identified largest and bottleneck links are redirected through lightly loaded paths to achieve better network load balancing. However, most of the existing scheme load balancing focuses on reducing network congestion; therefore, their methods heavily rely on the monitoring mechanism to obtain the flow of statistical information. Thus, the drawback of these schemes depends on the polling interval value to get the statistics. If a flow lasts longer than the statistics polling interval, obtaining such statistics at the regular interval requires an extra processing load. This process can potentially be resource-intensive for the SDN controller and introduce another scalability concern, mainly when the traffic flows increase and the number of switches to be monitored increases. Conversely, when the polling interval value is short, it increases the communication overhead switch–controller. Therefore, entirely depending on the controller to pull the data of total active flows to obtain the usage of the Flowtable may not be the best solution [36].

IV. FAILURE RECOVERY APPROACHES

Generally, failure management approaches can be divided into failure detection and recovery phases, each has its implementation mechanism and execution time. Failure recovery must be established immediately after link or switch failure detection. Afterwards, the recovery process is initiated. Most of the existing failure recovery proposals can be categorized based on their link, switch, or path recovery scope. This way, the configuration of the backup path could be per switch, link, or every disrupted flow using local and path protection or restoration mechanism. Other works focus on the hybridization of protection and restoration to achieve the maximum benefit of both. Intuitively, the recovery schemes are categorized into four (4) categories; This includes TCAM memory aware, load balancing aware, Quality of Service (QoS) aware, and rule update operation awareness based on either restoration, protection, or hybrid as presented in Fig. 4. This section discusses and analyses these approaches and highlight their weaknesses.

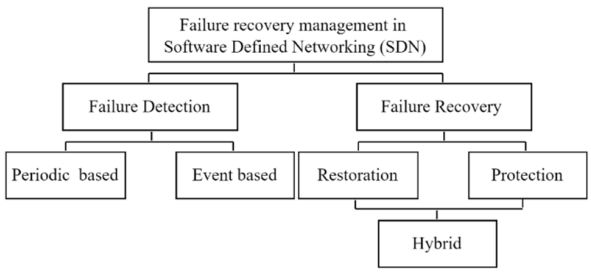

FIGURE 4. Failure recovery management taxonomy.

A. FAILURE DETECTION APPROACHES

The most crucial role of the SDN Controller is to maintain a real-time state and consistent network topology. Due to

TABLE 5. Failure detection approaches.

Approaches	Detection Schemes	Operation Mode	Limitation
[119]	LOS detects port down	Periodic	Status polling overhead
[114]	The lightweight software-based failure detection scheme	Periodic	Controller and switch overhead may not be suitable in large-scale network
[113]	Packet tagging/BER threshold monitoring using OpenFlow	Periodic	Frequent threshold value computation may introduce extra processing load on the controller
[112]	MPLS BFD Packet generators	-	A large number of packet generators consumed storage space
[110] [111]	BFD Exchange of echo packets messages	Periodic	A large number of echo packets overflow the Flowtable
[115] [120]	Heartbeat message	Periodic	Message overhead and storage space
[116]	Probe mechanism	Periodic	Probe packet overhead
[22]	Failure Detection based on event listening	Event	Longer detection time
[117]	Switch failure detection (SFD)	Periodic	The mechanism is limited to switch only

frequent network changes, links state changes more often. In this regard, the link discovery process is initiated to detect the link failure between connected OpenFlow switches and to efficiently detect changes to the network topology. Delay in link failure detection increases packet loss and increases failure recovery convergence time. Table 5 summarizes some of the existing detection schemes. Inefficient link failure detection can significantly affect the network operation that depends entirely on the SDN controller [109]. The fast failover group type was introduced in OpenFlow 1.1 to handle link failures locally without controller intervention. This can be achieved by configuring the failover predefinition of failure recovery policies on the OpenFlow devices to support forwarding behaviors that depend on the local states of OpenFlow switches. Toward this goal, references in [110], [111], and [112] leverage on failover scheme and use Bidirectional Forwarding Detection (BFD) to detect individual link failure. Each switch established a BFD session with its neighbor using a three-way handshake. Afterwards, the switches exchange echo messages to monitor the link state. If link failure occurs, the switch before the affected switches will detect the failure through an echo message and immediately communicate to the controller. In return, the controller instructs the affected switches to remove the link from its Flowtable.

In contrast, the work in [113] presents a link detection mechanism based on outgoing packets. The installed flow rules on the link were tagged and monitored, and packets were counted at the destination. Send packets are recorded, and the difference between sent and received packages is compared with the threshold value used to calculate the error rate. If the error rate is greater than a threshold value for each given link, the link is assumed to have failed. However, frequent threshold value computation may introduce extra processing load on the controller.

Other solutions used the concept of a monitoring cycle to detect failure location. The reference in [114] proposes a lightweight software-based failure detection scheme by exchanging alive packets between neighboring switches and controllers. Each switch that received the packet will create two copies, one to the controller and the other to the next switch. When the controller does not receive the packet from the expected switch for some time, it will be declared down. A similar concept was introduced in [115].

To some extent, BFD can help to detect failure quickly; however, this may be more applicable in a small-scale network. The presence of large applications may result in network traffic congestion, especially in a large-scale network. An alternative path needs to be established on time upon occurrences of failure; backup entries and BFD packets may easily congest the network and overflown the switch Flowtable. The proposal in [116] devised a centralized probe mechanism to detect link failures in the network. Even though, the minimum interval between probe packets is shown to have affected the failure recovery time. However, centralizing the failure detection may flood the network with probe packets, which can further exhaust the limited storage. Different from previously mentioned work. Reference [117] includes detection logic at the switch known as Switch Failure Detection (SFD). SFD detects failures by discovering the host connected to the switch and computing the packet loss ratio. If the loss ratio is 100 %, the switch is assumed to have failed else no failure. However, this scheme is limited to switch failure, and the frequency of link failure outweighs switch failure [118].

B. LOCAL AND PATH RESTORATION APPROACHES

Several proposals were made to configure the backup path for every disrupted flow [119], [121], as summarized in Table 6.

The works in [112] and [122] argued that periodic link monitoring to detect failure before establishing a backup path might considerably introduce controller overhead. To overcome these challenges [112], offload Operation Administration and Maintenance (OAM) link monitoring capability from the controller to the OpenFlow switch. OAM leverages on general message generator and processing function in the switches, and extension in OpenFlow 1.1 protocol to support the monitoring function. However, offloading some of the control functions to switches violates SDN's promise [123]. Alternatively, the approach in [124] presents a fault management scheme without modifying the SDN architecture. In this regard, a topology discovery module was devised to periodically collect the link state event. Afterwards, the route planning module used the gathered information to calculate multiple route paths based on the topology information. Upon failure, The VLAN switch configuration module configures multiple switch ports with relevant VLAN IDs to enforce each routing path. However, the scheme only focuses on recovering from failure, but consequences after a failure, such as potential failure or post-recovery congestion, were overlooked. A local reroute congestion-aware failure scheme was proposed in [106]. The scheme considered flows type and established path based on its requirement. When congestion occurs, re-routing is applied to the elephant while the mice flow packet forwarding continues. In this case, rerouting is performed locally at the point of congestion instead of re-diverting affected flows through available paths. However, triggering rerouting at a point of congestion can increase packet losses after failure. In addition, round trip time may also augment, affecting the failure convergence time. To overcome this limitation, the work in [125] devised a local fast reroute (LFR) to achieve faster recovery with less controller operation. After failure, all disrupted flows are aggregated into a new ''big'' flow. VLAN ID values are set to the aggregate flow; every packet is stamped with a new label and stored in VLAN ID. Thereafter, the local reroute path is dynamically deployed by the controller for the aggregated flows. In their follow-up work Cheng et al. [127] an integer linear programming model with heuristic model congestion awareness was introduced to avert link post-recovery congestion.

Similarly, the scheme in [128] introduced CAFFE to detour the affected flows as soon as the failure is detected to avoid potential congestion. Furthermore, CAFFE jointly considered knowledge of network topology, failure states, and network load distribution to formulate an Integer Linear Program (ILP) model to protect against potential future failure. However, their solutions heavily rely on aggregation and VLAN to perform fast rerouting, making computation expensive due to aggregation and may disenable the actual usage of VLAN [3].

Unlike the previously mentioned works, [120] argue that a reliable and scalable failure recovery scheme should minimise the controller's processing load and react even when the controller is not reachable. Toward these goals, the authors formulate a Mixed Integer Linear Programming Model (MILP) for precomputed backup recovery paths considering Quality of service (QoS) metrics. The scheme leverages on crank back signal to ensure instantaneous recovery times and aims at zero packet loss after failure detection, regardless of controller reachability, even when OpenFlow's ''fast-failover'' feature cannot be used. However, heavily relying on crank back routing may result in long backup paths and extra link usage. Their follow-up work SPIDER [115] implements the respective failure rerouting mechanism using MPLS tags. Furthermore, the scheme heavily depends on an extension of OpenFlow (Open state) to perform customized failure detection and data plane switching, making it incompatible with existing networks and available hardware switches [111].

In contrast, the previously mentioned works focus on reducing congestion, while others try reducing the controller's processing load. However, the authors consider the correlation between the switch Flowtable update operation and the recovery speed, which impacts the network convergence time. The reference in [125] presented Local Fast Reroute (LFR) flow aggregation techniques to a minimized number of flow operations. Once link failure occurs, the affected flows are aggregated into a single flow. Afterwards, the local reroute path will dynamically reroute the compressed flows using fast failover. Therefore, LFR can achieve faster failure recovery while minimizing flow operation. However, the merit depends on fast failover local reroute availability. Therefore, the availability of local reroute can affect the scheme convergence time. Moreover, LFR may also lead to larger packet drops when an end-to-end path is applied.

In contrast, [76] presented a failure restoration technique for minimizing the recovery cost. An ILP model was formulated to find a path with the lowest cost requiring up to a number of operations. Some path requires the fewest possible operations, and Dijkstra-like path cost requires minimum operations. The lowest possible operations, like that of the Dijkstra algorithm, are used to optimize the recovery time of failure. For all the sets of paths, a threshold value property was used to minimize the set of resulting paths. However, such a property may not always guarantee the feasibility of an end-to-end path [22]. An end-to-end fast link failure recovery approach based on the shortest was introduced in [117]. Packets are categorized into high and low-priority packets. When failure occurs, high-priority packets are rerouted through minimum delay. Other packets are forwarded through an alternative path; thus, the later and former traffic flows are distributed equally over the available paths to avoid congestion. However, the solution is tested in a small network setting, which may not be feasible in a large-scale network because the algorithm's complexity augments as the network's size increases [12]. The work of [129] proposed a mechanism to avoid frequent contacting the controller and take local corrective measures. In this regard, two methods were devised to store bypass paths on all pairs of nodes and others on some selected nodes. When a failure occurs,

TABLE 6. Summary of local and path restoration approaches.

Approaches	Aim	Scheme/ Techniques	Congestion Aware	Path Criticality	Limitation
[131]	Aimed to develop a fault-tolerant SDN architecture that can rapidly recover from faults and scale to large network sizes	CORONET mechanism	X	X	May introduce switch overhead of pull the monitoring information with extra storage
[112]	Aim to develop link monitoring without significant load on the controller	OAM mechanism	X	X	Offloading some control functions to switches violates SDN promised.
[106]	Aim to effectively manage congestion in the event of link congestion or failure	ILP model	√	X	Triggering the congestion scheme after the failure has occurred can significantly increase packet loss
[120][115]	Aims at reducing the processing load on the controller	MILP	X	X	The use of a crank back can lead to a longer backup. Open state is yet to be implemented as an extension of OpenFlow.
[76]	We aim to minimize the number of operations in such cases	ILP model	X	√	Path-based on threshold may not always guarantee the feasibility of an end-to-end path
[130][26][22]	To reduce the update operation to speed up recovery convergence time	Community detection scheme	X	X	Neither guarantees the shortest path and no post-recovery congestion leading to higher packet loss
[125]	reduces the number of flow operations between the SDN controller and switches	ILP model	X	X	local fast failover may not be available, or they may be inefficient from the resource allocation point of view
[129]	Aim at providing network resiliency when packets are in transit and a network link failure occurs	Path restoration model	X	X	Installing many bypass rules in the switch flowtable can lead to congestion which in turn increases packet loss
[126][125]	Aim to reduce the congestion after failure	ILP model	√	X	Depending on VLAN may cancel its actual usage. Aggregation can stop the controller flows visibility which in turn affects throughput
[117]	Finding an alternate path to handle high-priority packets with minimal delay when a link failure occurs in the network	Fast reroute network model	√	X	may not be feasible in a large-scale network, because the complexity of the algorithm augments as the network increase in size
[128]	Aim to avoid potential congestion in the post-recovery network	ILP model	√	X	Frequent changes in the network lead to the execution of the model more often and ILP required a fast solver to converge in a large-scale network.
[26]	To reduce the update operation to speed up recovery convergence time	Community detection scheme	X	X	Neither guarantees the shortest path and no post-recovery congestion leading to higher packet loss

the switch can act locally. However, installing two sets of flow rules bypass in the switch Flowtable beside the primary path rules will lead to load imbalance and cause congestion. Besides, this will further slow down the network reconfiguration because of the large number of updates.

Although restoration recovery schemes offer a more flexible way to handle verse flows in real-time, the cost of end-to-end computation is very high [26]. In this case, the restoration scheme is time-consuming as the SDN controller needs to calculate the new end-to-end path for each affected

flow and reconfigure the network. Therefore, the time to reconfigure the networks includes new path calculation and switch update time. To reduce these issues, [130] introduced the principle of a community detection scheme. If a failure occurs, the affected community is detected, and a backup path is established within the affected community without tampering with packet forwarding in other communities. This recovery is faster, thereby improving the network fault tolerance capability. However, removing the old flow entries in the affected path and re-installing the new entries for the alternative can be costly, especially when the path length is long. This concern has been addressed in their follow-up works [130], [26]. When failure occurs, the scheme only searches the new path from the point of failure down to the destination switch and removes the old flow entries of the affected switches only; the remaining flow entries on the path are preserved. Therefore, the scheme has significantly reduced the update operation and end-to-end computation time. However, the scheme neither guarantees the shortest path nor considers congestion.

C. LOCAL AND PATH PROTECTION APPROACHES

Failure recovery range can be divided into a path or local recovery. In the protection mechanism, rules must be preinstalled in advance for path and local recovery approaches. Due to the proliferation of network flows per second, the number of forwarding rules in the switch flow table may quickly escalate, leading to large switch memory TCAM consumption. To minimize the TCAM consumption, the references [7], [132] proposed a set of algorithms: Forward Local Re-routing(FLR) and Backward Local Rerouting (BLR) to compute backup paths for a primary path for faster failure recovery. Local re-routing of the failed traffic from the point of failure enables speedy recovery. FLR and BLR backup paths improved sharing of forwarding rules at the switches, thereby choosing a backup path with the least number of additional switches. However, the solution neither considers post-recovery congestion nor the potential future failure of the selected link. The references [133], [134], [135] leverage VLAN tagging to present a new protection method to aggregate the disrupted flows. This way, many flow rules for rerouting the affected flows are reduced, thereby improving the recovery time of carrier-grade recovery requirements. However, the aggregation technique may reduce flow visibility which in turn may be challenging for the controller to maintain the global view of the network. Their fellow up work implements two algorithms; Local Immediate (LIm) recovery strategy, in which the controller will utilize the fast failover to locally switch to an alternative path using the VLAN tagging feature to tag the arriving packets with the outgoing link ID. While Immediate Controller Dependent (ICoD) recovery required controller intervention to establish an alternative path. This way, recovery time may be faster because of the fast failover, which allows the quick and local reaction to failures without the need to resort to the central controller.

Several attempts have been made to achieve faster recovery using fast failover to avoid controller involvement in detouring the affected flows. Table 7 summarizes the related works. The reference in [136] proposed fast failover link failure with a congestion-aware mechanism. Fast failover is preconfigured with multiple paths to redirect the disrupted flows to a failure state. Based on the mechanism configuration, the controller periodically monitors the status of the switch port to perceive the failure on time. The protection scheme resulted in an average recovery time of around 40 ms. However, constant controller monitoring can introduce extra processing load on the controller. A scalable multi-failure fast failover was presented in [137]. Their work dynamically compressed the alternate path's flow entries of the incoming flows with the existing flow entries on the backup path. In this way, the total number of rules was significantly reduced. However, such a dynamic procedure may lead to extra processing load on the controller to configure the primary and backup path for every new arrival flow.

Moreover, the number of backup path rules augment as the flow arrival increases [138]. An efficient fault-tolerant memory management aware approach was proposed [133]. The scheme computes path protection per link instead of rules per flow by configuring VLAN tagging for each link failure. This way, A VLAN tagging is provided for each link identification while defining backup path rules. Therefore, the number of rules would be proportional to the network setting. However, in a large-scale network, the overhead would be non-trivial [9]. Another solution [139], [139] considered the switch Flowtable storage constraints to devise a recovery scheme. A Fault-Tolerant Forwarding Table Design (FFTD) was introduced to group the flows using group entries and aggregates the flows using a tagging mechanism for rapid recovery from the dual failures. This way, FFTD satisfies the GCN's 50 ms recovery requirement, reducing the backup path flow storage requirement. However, neither [133] nor [139] considered post-recovery congestion after the localized recovery. A shared ring was proposed as yet another solution to reduce the consumption of a backup resource [140]. The authors devised a ring-based single failure recovery approach to reduce the number of entries.

A ring circle in the network topology is selected to act as a share backup path, based on the all-backup path is introduced to improve the Flowtable utilization. Although recovery time and backup resource consumption could be improved, network post-recovery congestion may occur. Therefore, efficient congestion and memory-aware failure recovery (SafeGuad) were presented [118]. SafeGuard iterates through a backup path of the impacted flows to ensure that the rerouting switches have enough space to accommodate the backup path rules. In addition, residual link capacity is checked to avoid post-recovery congestion. This way, impacted flows are deployed efficiently. However, Safeguard may be expensive because it requires two paths to be installed for each flow, which could overwhelm critical network resources such as switch TCAM [26].

TABLE 7. Summary of local and path protection approaches.

Approaches	Aim	Techniques /scheme	Congestion aware	Path Criticality	Limitation
[132][7]	Aim to reduce the number of forwarding table entries with improved sharing of forwarding rules	ILP model	X	X	Post-recovery congestion may lead to larger packet loss
[133] [135]	Aim to reduce the number of rules to reroute the disrupted flows	LP model	X	X	Fast failover is used to define local detours. It may require controller intervention
[136]	To improve failover switch time and fast failure detection	Protection	√	X	Constant controller monitoring can introduce extra processing load on the controller.
[137]	Aim to provide enough backup routes within the available space in fast failover to address multiple failures	Protection	√	X	The procedure may lead to extra processing load on the controller to configure the primary and backup path for every new arrival flow
[133]	Aim to propose a protection recovery method that is fast with low memory consumption.	Protection	X	√	Controller overhead may increase as the network size increase
[142][143]	Aim to minimize the consumption of backup resources and meet the required failure recovery delay	Protection	X	√	Provisioning a double path may affect the system throughput
[139]	Aim to reduce the number of entries in fast failover for faster recovery	A fast failover recovery model	X	X	Fast failover local reroute may not be compactable with the OpenFlow switch
[140]	Aim to reduce the number of flow rules for faster failure recovery	A shared ring recovery model	X	X	The share ring procedure may lead to network post-recovery congestion may occur.
[141]	Aim to increase service availability and decrease disruption due to failure	Failure prediction model	X	√	Overlooks post-recovery congestion of the selected path, which may lead to more packet loss
[144]	Aims to introduce a recovery scheme to improve bandwidth allocation and switch-memory usage	MILP optimization model	√	X	ILP may require a fast solver to converge as the network and flows change more often. It may not be suitable for large-scale network

Most of the previously mentioned work heavily depends on fast failover local reroute to achieve faster failure recovery. Other solutions considered VLAN tagging with aggregation. Fast Failover (FF) local reroute approaches provide an efficient means of achieving fast failure recovery. FF can handle failure locally, thereby detouring the affected flows around the failed link using preconfigured alternative without the need to consult the controller. However, fast failover local reroute can only be used to define local detour mechanisms when alternative paths are available from the node that detects the failure. Therefore, the unavailability of a path may still require controller intervention, increasing the processing load.

Moreover, the FF group feature is optional in OpenFlow; these solutions' applicability depends on the FF group's actual hardware support [138]. Thus, several works have devised various recovery approaches to meet the recovery requirement. Other works attempt to reduce the number of flow entries due to many flow rules configured per switch in FF. However, decreasing service disruption, unavailability, and increasing availability to speed up the end-to-end convergence process time is being overlooked, unlike the existing works. For example, the work in [141] proposed Smart Routing (SR), which enabled the controller to receive early failure signs and avoid risky paths before the occurrence of the failure. SR predicts link failure events and formulates

an alternative path for some links. The risky path is a bypass to avoid future failure, leading to higher packet loss and decreased service availability. The mean time between failure and mean time to recover for evaluating the availability and reliability are considered. However, SR overlooks post-recovery congestion of the selected path, which may lead to more packet loss; the solution does not consider the system throughput or round trip time after failure convergence time.

D. HYBRID RESTORATION AND PROTECTION APPROACHES

Several attempts have been made lately to address the issues of failure, either restoration or protection. However, it is evident both approaches inherent some defects due to time and storage space gaps which need further investigation. Therefore, some solutions have been proposed to allocate backup rules flexibly by combining the restoration and protection modes to fulfil different application requirements without paying for their drawback, as summarized in Table 8. The work in [145] divided flows into Gold, Silver, and Bronze and provisioned different backup path strategies. Bronze flows provision backup paths reactively, while silver and bronze flow enjoyed proactive backup paths. The priorities of backup and the primary path of gold flows are the same. Gold flows are sent to the destination along two paths simultaneously. This way, better load balancing could be achieved. However, such a procedure may decline the system throughput and increase the chances of congestion, leading to switch Flowtable memory overflow. To optimize the Flowtable storage, [146] considered the switch Flowtable storage constraints to devise a recovery scheme. The authors derived link importance metrics and classified links based on the number of flows passing and bandwidth utilization ratio. Three backup strategies are introduced: double-path protection, single-path protection, and the reactive mode. Double path protection is deployed for high importance and links others through a single path. In this way, the limited switch storage resources are used to protect more essential links. However, provisioning a double path may affect the system throughput. Motivated by [147], [146], their recovery approaches considered delay during back path selection and flow classification received little attention. The impact of failure varies with respect to flows, and losses caused by failure differ from other flows. In this regard, efficient failure recovery is required to consider flows. This way, the authors formulate a formula to derive link importance and provision backup path based on flow importance. Neither [147] nor [146] consider post-recovery congestion nor potential failure as the result of the selected path.

In contrast, Revive [148] was introduced to proactively install alternative routes on a subset of switches between a given source-designation pair. Other switches are installed backup rules along with primary reactively on demand. However, this approach requires additional path computation for backup paths along the primary path, and the implication of the selected path was overlooked.

A flexible recovery mechanism (BOND) was presented [9] to deal with a link failure. BOND preconfigured backup path rules in the switch Flowtable to establish an alternative path in case of failure. A Hash table was used to quickly recover link failure. However, if a switch fails to operate, the system will lose all the routing path information of the network. References [149], [150] present another solution for efficient and flexible link fault tolerance (FTLink). A collection of the backup path was generated for primary links. This way, FTLink maintained a matching table for maintaining an alternative path entries at the controller. When the system detects failures, it enables the backup path link as the new primary link for the affected link. Flow rules are installed in the switch after they are matched from the generated table using the controller. However, FTLink overlooks a link's criticality before enabling the backup path link as the new primary link for the affected [150]. Such a procedure may lead to future failure since the backup path may fail earlier than the primary path.

V. RESEARCH CHALLENGES AND FUTURE DIRECTION FOR DYNAMIC ROUTING AND FAILURE RECOVERY

Existing studies have proposed different routing and failure recovery solutions for various use cases. However, dynamic routing based on applications requirement and efficient failure recovery while managing resource utilization is still a challenging research area with several unanswered questions. This section discussed some of the unaddressed challenges and suggested future research direction.

A. OPTIMIZED ENERGY-AWARE ROUTING WITH SWITCH UPDATE OPERATION

Most of the existing Energy-Aware Routing focuses on topology awareness, queue engineering, or smart sleep-on strategies to optimize network resource energy consumption during the routing process. However, update operation in SDN switch TCAM is very slow, while traffic flow changes more often with the large number of flows arriving in a dynamic large-scale network. This process affects the switch updating time with significant processing delay besides power consumption. It would be an interesting research direction to devise a dynamic energy-aware routing while incorporating switch TCAM update operation and link processing power

B. DYNAMIC ROUTING WITH QOS AWARE

Traffic flows exhibit variabilities with different Quality of Service requirements. Some flows are delay-sensitive, others require sufficient link bandwidth, throughput sensitive flows are among flows with special needs. Some flows required multiple routing constraints for optimal performance. Similarly, routing paths differ, some paths are very critical with limited bandwidth, and others have smaller path latency. It is

TABLE 8. Summary of hybrid recovery approaches.

Approaches	Aim	Schemes/ Techniques	Congestion	Path criticality	Limitation
[155]	Achieved better recovery with load balancing	Path computation and traffic recovery model	X	X	The procedure may affect the system through an increased congestion
[156]	To reduce the consumption of backup resources, faster failure recovery	Link classification model	X	√	Provisioning a double path may affect the system throughput and increase the chance of congestion
[157]	Aim to classify flows and link to provision efficient backup path	Flows classification and link performance metric model	X	√	Potential failures as the result of the chosen path were overlooked.
[158]	Hybrid recovery approach for faster recovery time and low number of entries	Revive recovery model	X	X	Post-recovery congestion was overlooked, consequently increasing packet loss
[9]	To devise a flexible backup strategy with reactive and proactive	ILP	√	X	ILP may require a fast solver to converge as the network and flows change more often. It may not be suitable in large-scale net
[159]	Aim to reduce the number of entries for quicker recovery	ILP Heuristic model	√	X	

challenging to find a path with both properties in both terms. Although several dynamic routings have been presented, they overlooked incorporating multiple routing metrics for optimal routing. Therefore, how to aggregate various routing metrics using fuzzy logic while efficiently utilizing the limited Flowtable resource is still an open question.

C. POST-RECOVERY CONGESTION AND INCREASE IN SERVICE DISRUPTION

Restoration recovery schemes offered more flexibility to cope with frequent traffic changes. However, frequent path computation may impose an extra processing load on the controller, and the time to update switches may also introduce another bottleneck. This way, several solutions were proposed to address these issues. Some solutions focus on reducing the computational path time, and others introduced path selection costing a certain number of operations. Failure recovery with congestion was also proposed. However, these solutions may not always ensure full service is available and decrease disruption. In large-scale networks, path importance differs; some paths are critical because of the shortest paths that pass through them. Other links on paths are frequently shared between the primary and backup paths. Therefore, failures on either path may lead to failure on multiple paths. Consequently, service disruption leads to significant packet losses and a decline in throughput. Therefore, an efficient recovery scheme with congestion awareness and path reliability considering real traffic flows is required to improve service availability and decrease system disruption with post-recovery congestion awareness

D. HYBRID RESTORATION AND PROTECTION RECOVERY APPROACH

Both restoration and protection have their pros and cons. Therefore, combining the two approaches for efficient failure recovery while meeting the requirement of different flows would be another interesting research work. There exist different flows with different Quality of Service requirements. Packets from video traffic will be required to be redirected through a path with sufficient bandwidth to accommodate many affected flows upon failure occurrences. While delay-sensitive flows such as VoIP may require paths with small delays toward a destination. This way, it would be interesting research to classify flows based on their quality of service requirement and flexibly apply restoration and protection without paying for their drawback.

VI. CONCLUSION

Software Defined Network is an emerging network with better network management. It speeds up network innovation. However, it has some weaknesses which require urgent attention to speed up the adoption of SDN. Several researchers have been conducted over the years to address different challenges introduced by SDN. Route path selection and failure recovery are among the challenges affecting the SDN. This paper presents A survey for dynamic routing and failure recovery approaches for efficient resource utilization. The paper elaborates on the concepts and fundamental knowledge of Software-defined Networks required to efficiently design and implement reliable failure recovery. A comprehensive review of path route selection toward better failure recovery

schemes was reviewed. Tables were presented to support the study, and critical evaluations of the existing schemes were discussed to highlight the weakness of the existing schemes. To overcome the limitation of the existing schemes, this research suggests a future direction to enhance the network performance of the existing literature with efficient usage of the switch Flowtable and less overhead.

REFERENCES

[1] M. Alsaeedi, M. M. Mohamad, and A. A. Al-Roubaiey, "Toward adaptive and scalable OpenFlow-SDN flow control: A survey," *IEEE Access*, vol. 7, pp. 107346–107379, 2019.

[2] Y. Yu, X. Li, X. Leng, L. Song, K. Bu, Y. Chen, J. Yang, L. Zhang, K. Cheng, and X. Xiao, "Fault management in software-defined networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 349–392, 2019, doi: 10.1109/COMST.2018.2868922.

[3] A. Malik, B. Aziz, A. Al-Haj, and M. Adda, "Software-defined networks: A walkthrough guide from occurrence To data plane fault tolerance," *PeerJ Preprints*, vol. 7, p. e27624v1, 2019, doi: 10.7287/peerj.preprints.27624v1.

[4] J. J. Garcia-Luna-Aceves, "A minimum-hop routing algorithm based on distributed information," *Comput. Netw. ISDN Syst.*, vol. 16, no. 5, pp. 367–382, 1989, doi: 10.1016/0169-7552(89)90011-1.

[5] L. Chen, B. Li, and B. Li, "Barrier-aware max-min fair bandwidth sharing and path selection in datacenter networks," in *Proc. IEEE Int. Conf. Cloud Eng. (ICE)*, Apr. 2016, pp. 151–160, doi: 10.1109/IC2E.2016.35.

[6] S. Kotachi, T. Sato, R. Shinkuma, and E. Oki, "Multicast routing model to minimize number of flow entries in software-defined network," *IEICE Trans. Commun.*, vol. 104, no. 5, pp. 507–518, May 2021, doi: 10.1587/TRANSCOM.2020EBP3064.

[7] P. M. Mohan, T. Truong-Huu, and M. Gurusamy, "Fault tolerance in TCAM-limited software defined networks," *Comput. Netw.*, vol. 116, pp. 47–62, Apr. 2017, doi: 10.1016/j.comnet.2017.02.009.

[8] A. S. Alshra'a, P. Sewalkar, and J. Seitz, "Enhanced failure recovery mechanism using OpenState pipeline in SDN," in *Proc. 10th Int. Conf. Ubiquitous Future Netw. (ICUFN)*, Jul. 2018, pp. 104–109, doi: 10.1109/ICUFN.2018.8437006.

[9] Q. Li, Y. Liu, Z. Zhu, H. Li, and Y. Jiang, "BOND: Flexible failure recovery in software defined networks," *Comput. Netw.*, vol. 149, pp. 1–12, Feb. 2019, doi: 10.1016/j.comnet.2018.11.020.

[10] P. C. D. R. Fonseca and E. S. Mota, "A survey on fault management in software-defined networks," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2284–2321, 4th Quart., 2017, doi: 10.1109/comst.2017.2719862.

[11] 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.

[12] 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, p. 4255, 2020.

[13] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[14] T. Kato, M. Kawakami, T. Myojin, H. Ogawa, K. Hirono, and T. Hasegawa, "Case study of applying SPLE to development of network switch products," in *Proc. 17th Int. Softw. Product Line Conf.*, 2013, pp. 198–207, doi: 10.1145/2491627.2491636.

[15] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "OpenState: Programming platform-independent stateful openflow applications inside the switch," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 44–51, 2014, doi: 10.1145/2602204.2602211.

[16] M. Kodialam and T. V. Lakshman, "Minimum interference routing with applications to MPLS traffic engineering," in *Proc. IEEE INFOCOM Conf. Comput. Commun., 19th Annu. Joint Conf. IEEE Comput. Commun. Societies*, Mar. 2000, pp. 884–893.

[17] Y.-R. Chiang, C.-H. Ke, Y.-S. Yu, Y.-S. Chen, and C.-J. Pan, "A multipath transmission scheme for the improvement of throughput over SDN," in *Proc. Int. Conf. Appl. Syst. Innov. (ICASI)*, May 2017, pp. 1247–1250, doi: 10.1109/ICASI.2017.7988122.

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

[19] E. Akin and T. Korkmaz, "Comparison of routing algorithms with static and dynamic link cost in software defined networking (SDN)," *IEEE Access*, vol. 7, pp. 148629–148644, 2019, doi: 10.1109/ACCESS.2019.2946707.

[20] W. Jiawei, Q. Xiuquan, and N. Guoshun, "Dynamic and adaptive multi-path routing algorithm based on software-defined network," *Int. J. Distrib. Sensor Netw.*, vol. 14, no. 10, Oct. 2018, Art. no. 155014771880568, doi: 10.1177/1550147718805689.

[21] R. Li and X. Wang, "A tale of two (flow) tables: Demystifying rule caching in OpenFlow switches," in *Proc. 48th Int. Conf. Parallel Process.*, 2019, pp. 1–10.

[22] 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.

[23] A. Malik, B. Aziz, and M. Bader-El-Den, "Finding most reliable paths for software defined networks," in *Proc. 13th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, 2017, pp. 1309–1314.

[24] K. Qiu, J. Yuan, J. Zhao, X. Wang, S. Secci, and X. Fu, "FastRule: Efficient flow entry updates for TCAM-based OpenFlow switches," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 484–498, Mar. 2019, doi: 10.1109/JSAC.2019.2894235.

[25] Y. Wang, S. Feng, H. Guo, X. Qiu, and H. An, "A single-link failure recovery approach based on resource sharing and performance prediction in SDN," *IEEE Access*, vol. 7, pp. 174750–174763, 2019, doi: 10.1109/ACCESS.2019.2957141.

[26] A. Malik, R. D. Fréin, and B. Aziz, "Rapid restoration techniques for software-defined networks," *Appl. Sci.*, vol. 10, no. 10, p. 3411, May 2020, doi: 10.3390/app10103411.

[27] T. Hu, P. Yi, Z. Guo, J. Lan, and Y. Hu, "Dynamic slave controller assignment for enhancing control plane robustness in software-defined networks," *Future Gener. Comput. Syst.*, vol. 95, pp. 681–693, Jun. 2019, doi: 10.1016/j.future.2019.01.010.

[28] S. Dou, Z. Guo, and Y. Xia, "ProgrammabilityMedic: Predictable path programmability recovery under multiple controller failures in SD-WANs," in *Proc. IEEE 41st Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2021, pp. 461–471, doi: 10.1109/ICDCS51616.2021.00051.

[29] S. Dou, G. Miao, Z. Guo, C. Yao, W. Wu, and Y. Xia, "Matchmaker: Maintaining network programmability for software-defined WANs under multiple controller failures," *Comput. Netw.*, vol. 192, Jun. 2021, Art. no. 108045, doi: 10.1016/j.comnet.2021.108045.

[30] Z. Guo, S. Dou, and W. Jiang, "Improving the path programmability for software-defined WANs under multiple controller failures," in *Proc. IEEE/ACM 28th Int. Symp. Quality Service (IWQoS)*, Jun. 2020, pp. 1–10.

[31] X.-N. Nguyen, D. Saucez, C. Barakat, and T. Turletti, "Rules placement problem in OpenFlow networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 2, pp. 1273–1286, 2nd Quart., 2016, doi: 10.1109/COMST.2015.2506984.

[32] X. Nguyen, D. Saucez, C. Barakat, and T. Turletti, "OFFICER: A general optimization framework for OpenFlow rule allocation and endpoint policy enforcement," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 478–486, doi: 10.1109/INFOCOM.2015.7218414.

[33] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "CacheFlow: Dependency-aware rule-caching for software-defined networks categories and subject descriptors," in *Proc. Symp. SDN Res.*, 2016, pp. 1–12.

[34] M. Rifai, N. Huin, C. Caillouet, F. Giroire, J. Moulierac, D. L. Pacheco, and G. Urvoy-Keller, "Minnie: An SDN world with few compressed forwarding rules," *Comput. Netw.*, vol. 121, pp. 185–207, Jul. 2017, doi: 10.1016/j.comnet.2017.04.026.

[35] B. Isyaku, M. B. Kamat, K. B. A. Bakar, M. S. M. Zahid, and F. A. Ghaleb, "IHTA: Dynamic idle-hard timeout allocation algorithm based OpenFlow switch," in *Proc. IEEE 10th Symp. Comput. Appl. Ind. Electron. (ISCAIE)*, Apr. 2020, pp. 170–175, doi: 10.1109/ISCAIE47305.2020.9108803.

[36] B. Isyaku, K. A. Bakar, M. Soperi, and M. Zahid, "Adaptive and hybrid idle–hard timeout allocation and flow eviction mechanism considering traffic characteristics," *Electronics*, vol. 9, no. 11, p. 1983, 2020.

[37] L. Zhang, S. Wang, S. Xu, R. Lin, and H. Yu, "TimeoutX: An adaptive flow table management method in software defined networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2015, pp. 1–6, doi: 10.1109/GLOCOM.2015.7417563.

[38] U. Humayun, M. Hamdan, and M. N. Marsono, "Early flow table eviction impact on delay and throughput in software-defined networks," in *Proc. 11th IEEE Int. Conf. Control Syst., Comput. Eng. (ICCSCE)*, Aug. 2021, pp. 27–28.

[39] D. Todorov, H. Valchanov, and V. Aleksieva, "Simple routing algorithm with link discovery between source and destination hosts in SDN networks," in *Proc. Int. Conf. Autom. Informat. (ICAI)*, Sep. 2021, pp. 188–191, doi: 10.1109/ICAI52893.2021.9639742.

[40] P. Sun, Z. Guo, J. Lan, J. Li, Y. Hu, and T. Baker, "ScaleDRL: A scalable deep reinforcement learning approach for traffic engineering in SDN with pinning control," *Comput. Netw.*, vol. 190, May 2021, Art. no. 107891, doi: 10.1016/j.comnet.2021.107891.

[41] J. Zhang, M. Ye, Z. Guo, C.-Y. Yen, and H. J. Chao, "CFR-RL: Traffic engineering with reinforcement learning in SDN," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 10, pp. 2249–2259, Oct. 2020, doi: 10.1109/JSAC.2020.3000371.

[42] J. Zhang, Z. Guo, M. Ye, and H. J. Chao, "SmartEntry: Mitigating routing update overhead with reinforcement learning for traffic engineering," in *Proc. Workshop Netw. Meets AI ML*, Aug. 2020, pp. 1–7, doi: 10.1145/3405671.3405809.

[43] P. Sun, J. Li, Z. Guo, Y. Xu, J. Lan, and Y. Hu, "SINET: Enabling scalable network routing with deep reinforcement learning on partial nodes," in *Proc. ACM SIGCOMM Conf. Posters*, 2019, pp. 88–89, doi: 10.1145/3342280.3342317.

[44] P. Sun, Z. Guo, S. Liu, J. Lan, J. Wang, and Y. Hu, "SmartFCT: Improving power-efficiency for data center networks with deep reinforcement learning," *Comput. Netw.*, vol. 179, Oct. 2020, Art. no. 107255, doi: 10.1016/j.comnet.2020.107255.

[45] H. Ni, Z. Guo, C. Li, S. Dou, C. Yao, and T. Baker, "Network coding-based resilient routing for maintaining data security and availability in software-defined networks," *J. Netw. Comput. Appl.*, vol. 205, Sep. 2022, Art. no. 103372, doi: 10.1016/j.jnca.2022.103372.

[46] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, no. 6, p. 345, Jun. 1962, doi: 10.1145/367766.368168.

[47] G. Apostolopoulos, "On the effectiveness of path pre-computation in reducing the processing cost of on-demand QoS path computation," in *Proc. 3rd IEEE Symp. Comput. Commun.*, Jun. 1998, pp. 42–46.

[48] Z. Wang and J. Crowcroft, "Routing algorithms for supporting resource reservation," *IEEE JSAC*, vol. 8, no. 3, pp. 368–379, Apr. 1990.

[49] O. A. Raouf and H. Askr, "ACOSDN-ant colony optimization algorithm for dynamic routing in software defined networking," in *Proc. 14th Int. Conf. Comput. Eng. Syst. (ICCES)*, 2019, pp. 141–148.

[50] N. A. El-Hefnawy, O. A. Raouf, and H. Askr, "Dynamic routing optimization algorithm for software defined networking," *Comput., Mater. Continua*, vol. 70, no. 1, pp. 1349–1362, 2022, doi: 10.32604/cmc.2022.017787.

[51] M. Huang, W. Liang, Z. Xu, W. Xu, S. Guo, and Y. Xu, "Dynamic routing for network throughput maximization in software-defined networks," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9, doi: 10.1109/INFOCOM.2016.7524613.

[52] S. Tomovic, I. Radusinovic, and N. Prasad, "Performance comparison of QoS routing algorithms applicable to large-scale SDN networks," in *Proc. IEEE Int. Conf. Comput. Tool (EUROCON)*, Sep. 2015, pp. 1–6, doi: 10.1109/EUROCON.2015.7313698.

[53] S. Tomovic and I. Radusinovic, "Fast and efficient bandwidth-delay constrained routing algorithm for SDN networks," in *Proc. IEEE NetSoft Conf. Workshops (NetSoft)*, Jun. 2016, pp. 303–311, doi: 10.1109/NETSOFT.2016.7502426.

[54] Q. Ma and P. Steenkiste, "On path selection for traffic with bandwidth guarantees," in *Proc. Int. Conf. Netw. Protocols*, 1997, pp. 191–202.

[55] L.-W. Cheng and S.-Y. Wang, "Application-aware SDN routing for big data networking," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–6, doi: 10.1109/glocom.2015.7417577.

[56] R. C. Ramirez, Q. T. Vien, R. Trestian, L. Mostarda, and P. Shah, "Multi-path routing for mission critical applications in software-defined networks," in *Proc. Int. Conf. Ind. Netw. Intell. Syst.* (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering), vol. 257, 2019, pp. 38–48, doi: 10.1007/978-3-030-05873-9_4.1

[57] A. Chooprateep, "Video path selection for traffic engineering in SDN," in *Proc. 11th Int. Conf. Inf. Technol. Elect. Eng. (ICITEE)*, vol. 7, 2019, pp. 1–6.

[58] M. Beshley, M. Seliuchenko, O. Panchenko, and A. Polishuk, "Adaptive flow routing model in SDN," in *Proc. 14th Int. Conf. Exper. Designing Appl. CAD Syst. Microelectron. (CADSM)*, 2017, pp. 298–302, doi: 10.1109/CADSM.2017.7916140.

[59] M. F. Rangkuty and M. H. A. Al-hooti, "Path selection in software defined network data plane using least loaded path," in *Proc. Int. Conf. Adv. Comput. Sci. Inf. Syst. (ICACSIS)*, 2020, pp. 135–140.

[60] B. Isyaku, K. B. A. Bakar, F. A. Ghaleb, and M. S. M. Zahid, "Path selection with critical switch-aware for software defined networking," in *Proc. IEEE Symp. Wireless Technol. Appl. (ISWTA)*, Aug. 2021, pp. 22–26.

[61] B. Isyaku, K. A. Bakar, M. S. M. Zahid, E. H. Alkhammash, F. Saeed, and F. A. Ghaleb, "Route path selection optimization scheme based link quality estimation and critical switch awareness for software defined networks," *Appl. Sci.*, vol. 11, no. 19, p. 9100, Sep. 2021.

[62] S. Otoum, B. Kantarci, and H. Mouftah, "Empowering reinforcement learning on big sensed data for intrusion detection," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–7.

[63] K. B. Nougnanke, M. Bruyere, and Y. Labit, "Low-overhead near-real-time flow statistics collection in SDN," in *Proc. 6th IEEE Int. Conf. Netw. Softwarization (NetSoft)*, Ghent, Belgium, 2020, pp. 155–159.

[64] C. Pemer and G. Carle, "Comparison of optimization goals for resilient routing," in *Proc. IEEE Int. Conf. Commun. Workshops*, May 2019, pp. 1–6.

[65] M. Al-fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. NSDI*, 2010, pp. 89–92.

[66] B. Yang, X. Cao, J. Bassey, X. Li, T. Kroecker, and L. Qian, "Computation offloading in multi-access edge computing networks: A multi-task learning approach," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–6.

[67] H. Zhang, F. Tang, and L. Barolli, "Efficient flow detection and scheduling for SDN-based big data centers," *J. Ambient Intell. Humanized Comput.*, vol. 10, no. 5, pp. 1915–1926, May 2019, doi: 10.1007/s12652-018-0783-6.

[68] Z. Guo, Y. Xu, M. Cello, and J. Zhang, "JumpFlow: Reducing flow table usage in software-defined networks," *Comput. Netw.*, vol. 92, pp. 300–315, Dec. 2015, doi: 10.1016/j.comnet.2015.09.030.

[69] Z. Guo, Y. Xu, R. Liu, A. Gushchin, K.-Y. Chen, A. Walid, and H. J. Chao, "Balancing flow table occupancy and link utilization in software-defined networks," *Future Gener. Comput. Syst.*, vol. 89, pp. 213–223, Dec. 2018, doi: 10.1016/j.future.2018.06.011.

[70] L. Zhang, Q. Deng, Y. Su, and Y. Hu, "A box-covering-based routing algorithm for large-scale SDNs," *IEEE Access*, vol. 5, pp. 4048–4056, 2017, doi: 10.1109/ACCESS.2017.2682501.

[71] J. Dong, C. Ma, W. Cheng, and L. Xin, "Notice of violation of IEEE publication principles: Data augmented design: Urban planning and design in the new data environment," in *Proc. IEEE 2nd Int. Conf. Big Data Anal. (ICBDA)*, Mar. 2017, pp. 508–512.

[72] M. M. Mulla and S. Shinde, "Ant colony optimization-based dynamic routing in software defined networks," in *Proc. 11th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT)*, 2020, pp. 1–7.

[73] O. Dobrijevic, M. Santl, and M. Matijasevic, "Ant colony optimization for QoE-centric flow routing in software-defined networks," in *Proc. 11th Int. Conf. Netw. Service Manag. (CNSM)*, Nov. 2015, pp. 274–278, doi: 10.1109/CNSM.2015.7367371.

[74] Z. Guo, R. Liu, Y. Xu, A. Gushchin, A. Walid, and H. J. Chao, "STAR: Preventing flow-table overflow in software-defined networks," *Comput. Netw.*, vol. 125, pp. 15–25, Oct. 2017, doi: 10.1016/j.comnet.2017.04.046.

[75] S. Astaneh and S. S. Heydari, "Multi-failure restoration with minimal flow operations in software defined networks," in *Proc. 11th Int. Conf. Design Reliable Commun. Netw. (DRCN)*, Mar. 2015, pp. 263–266, doi: 10.1109/DRCN.2015.7149024.

[76] S. A. Astaneh and S. S. Heydari, "Optimization of SDN flow operations in multi-failure restoration scenarios," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 421–432, Sep. 2016, doi: 10.1109/TNSM.2016.2580590.

[77] S. M. Raza, S. Ahvar, R. Amin, and M. Hussain, "Reliability aware multiple path installation in software-defined networking," *Electronics*, vol. 10, no. 22, p. 2820, Nov. 2021, doi: 10.3390/electronics10222820.

[78] S. Kotachi, T. Sato, R. Shinkuma, and E. Oki, "Multicast routing model to minimize number of flow entries in software-defined network," in *Proc. 20th Asia–Pacific Netw. Oper. Manag. Symp. (APNOMS)*, 2019, pp. 1–6.

[79] R. Biswas and J. Wu, "Minimizing the number of rules to mitigate link congestion in SDN-based datacenters," in *Proc. IEEE Int. Conf. Netw., Archit. Storage (NAS)*, Oct. 2021, pp. 1–8, doi: 10.1109/NAS51552.2021.9605365.

[80] L. Luo, H. Yu, S. Luo, M. Zhang, and S. Yu, "Achieving fast and lightweight SDN updates with segment routing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–6, doi: 10.1109/GLOCOM.2016.7841562.

[81] H. Xu, Z. Yu, X. Y. Li, L. Huang, C. Qian, and T. Jung, "Joint route selection and update scheduling for low-latency update in SDNs," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 3073–3087, Oct. 2017, doi: 10.1109/TNET.2017.2717441.

[82] H. Xu, Z. Yu, X.-Y. Li, C. Qian, L. Huang, and T. Jung, "Real-time update with joint optimization of route selection and update scheduling for SDNs," in *Proc. IEEE 24th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2016, pp. 1–10, doi: 10.1109/ICNP.2016.7784436.

[83] M. Rifai, N. Huin, C. Caillouet, F. Giroire, J. Moulierac, D. L. Pacheco, and G. Urvoy-Keller, "MINNIE: An SDN world with few compressed forwarding rules," *Comput. Netw.*, vol. 121, pp. 185–207, Jul. 2017, doi: 10.1016/j.comnet.2017.04.026.

[84] Z. Zhao, W. Yang, and B. Wu, "Flow aggregation through dynamic routing overlaps in software defined networks," *Comput. Netw.*, vol. 176, Jul. 2020, Art. no. 107293, doi: 10.1016/j.comnet.2020.107293.

[85] T. Chao and K. Wang, "In-switch dynamic flow aggregation in software defined networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2017, pp. 1–6.

[86] F. Giroire, J. Moulierac, and T. K. Phan, "Optimizing rule placement in software-defined networks for energy-aware routing," in *Proc. IEEE Global Commun. Conf.*, Dec. 2014, pp. 2523–2529, doi: 10.1109/GLOCOM.2014.7037187.

[87] F. Giroire, N. Huin, J. Moulierac, and T. K. Phan, "Energy-aware routing in software-defined network using compression," *Comput. J.*, vol. 61, no. 10, pp. 1537–1556, Oct. 2018, doi: 10.1093/comjnl/bxy029.

[88] J. Galán-Jiménez, M. Polverini, and A. Cianfrani, "Reducing the reconfiguration cost of flow tables in energy-efficient software-defined networks," *Comput. Commun.*, vol. 128, pp. 95–105, Sep. 2018, doi: 10.1016/j.comcom.2018.07.022.

[89] H. Zhu, X. Liao, C. de Laat, and P. Grosso, "Joint flow routing-scheduling for energy efficient software defined data center networks: A prototype of energy-aware network management platform," *J. Netw. Comput. Appl.*, vol. 63, pp. 110–124, Mar. 2016, doi: 10.1016/j.jnca.2015.10.017.

[90] B. G. Assefa and Ö. Özkasap, "A survey of energy efficiency in SDN: Software-based methods and optimization models," *J. Netw. Comput. Appl.*, vol. 137, pp. 127–143, Jul. 2019, doi: 10.1016/j.jnca.2019.04.001.

[91] T. M. Nam, N. H. Thanh, N. Q. Thu, H. T. Hieu, and S. Covaci, "Energy-aware routing based on power profile of devices in data center networks using SDN," in *Proc. 12th Int. Conf. Electr. Eng./Electron., Comput., Telecommun. Inf. Technol. (ECTI-CON)*, Jun. 2015, pp. 1–6, doi: 10.1109/ECTICon.2015.7207042.

[92] G. Xu, B. Dai, B. Huang, and J. Yang, "Bandwidth-aware energy efficient routing with SDN in data center networks," in *Proc. IEEE 17th Int. Conf. High Perform. Comput. Commun., IEEE 7th Int. Symp. Cyberspace Safety Secur., 2015 IEEE 12th Int. Conf. Embedded Softw. Syst.*, Aug. 2015, pp. 766–771, doi: 10.1109/HPCC-CSS-ICESS.2015.12.

[93] G. Xu, B. Dai, B. Huang, J. Yang, and S. Wen, "Bandwidth-aware energy efficient flow scheduling with SDN in data center networks," *Future Gener. Comput. Syst.*, vol. 68, pp. 163–174, Mar. 2017, doi: 10.1016/j.future.2016.08.024.

[94] A. Amokrane, R. Langar, R. Boutaba, and G. Pujolle, "Flow-based management for energy efficient campus networks," *IEEE Trans. Netw. Service Manag.*, vol. 12, no. 4, pp. 565–579, Dec. 2015, doi: 10.1109/TNSM.2015.2501398.

[95] M. N. Siraj, N. Javaid, Q. Shafi, Z. Ahmed, U. Qasim, and Z. A. Khan, "Energy aware dynamic routing using SDN for a campus network," in *Proc. 19th Int. Conf. Network-Based Inf. Syst. (NBiS)*, Sep. 2016, pp. 226–230, doi: 10.1109/NBiS.2016.80.

[96] D. Jiang, P. Zhang, Z. Lv, and H. Song, "Energy-efficient multi-constraint routing algorithm with load balancing for smart city applications," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 1437–1447, Dec. 2016, doi: 10.1109/JIOT.2016.2613111.

[97] S. Hur, Y. J. Cho, J. Lee, N. G. Kang, J. Park, and H. Benn, "Synchronous channel sounder using horn antenna and indoor measurements on 28 GHz," in *Proc. IEEE Int. Black Sea Conf. Commun. Netw. (BlackSeaCom)*, May 2014, pp. 83–87.

[98] B. Ozbek, Y. Aydogmus, A. Ulas, B. Gorkemli, and K. Ulusoy, "Energy aware routing and traffic management for software defined networks," in *Proc. IEEE NetSoft Conf. Workshops (NetSoft)*, Jun. 2016, pp. 73–77, doi: 10.1109/NETSOFT.2016.7502446.

[99] R. Maaloul, R. Taktak, L. Chaari, and B. Cousin, "Energy-aware routing in carrier-grade Ethernet using SDN approach," *IEEE Trans. Green Commun. Netw.*, vol. 2, no. 3, pp. 844–858, Sep. 2018, doi: 10.1109/TGCN.2018.2832658.

[100] S. Torkzadeh, H. Soltanizadeh, and A. A. Orouji, "Energy-aware routing considering load balancing for SDN: A minimum graph-based ant colony optimization," *Cluster Comput.*, vol. 24, no. 3, pp. 2293–2312, Sep. 2021, doi: 10.1007/s10586-021-03263-x.

[101] Z. Guo, Y. Xu, Y.-F. Liu, S. Liu, H. J. Chao, Z.-L. Zhang, and Y. Xia, "AggreFlow: Achieving power efficiency, load balancing, and quality of service in data center networks," *IEEE/ACM Trans. Netw.*, vol. 29, no. 1, pp. 17–33, Feb. 2021, doi: 10.1109/TNET.2020.3026015.

[102] Z. Guo, S. Dou, Y. Wang, S. Liu, W. Feng, and Y. Xu, "HybridFlow: Achieving load balancing in software-defined WANs with scalable routing," *IEEE Trans. Commun.*, vol. 69, no. 8, pp. 5255–5268, Aug. 2021, doi: 10.1109/TCOMM.2021.3074500.

[103] S. M. Park, S. Ju, and J. Lee, "Efficient routing for traffic offloading in software-defined network," *Proc. Comput. Sci.*, vol. 34, pp. 674–679, Aug. 2014, doi: 10.1016/j.procs.2014.07.096.

[104] C. N. Sminesh and K. Ranjitha, "A proactive flow admission and re-routing scheme for load balancing and mitigation of congestion propagation in SDN data plane," *Int. J. Comput. Netw. Commun.*, vol. 10, no. 6, pp. 117–134, Nov. 2018, doi: 10.5121/ijcnc.2018.10607.

[105] J. Hwang, J. Yoo, S.-H. Lee, and H.-W. Jin, "Scalable congestion control protocol based on SDN in data center networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2015, pp. 1–6.

[106] R. Kanagevlu and K. M. M. Aung, "SDN controlled local re-routing to reduce congestion in cloud data center," in *Proc. Int. Conf. Cloud Comput. Res. Innov. (ICCCRI)*, Oct. 2015, pp. 80–88, doi: 10.1109/ICCCRI.2015.27.

[107] M. Kao, B. Huang, S. Kao, and H. Tseng, "An effective routing mechanism for link congestion avoidance in software-defined networking get switches information record switches port translate to graph wait 3 seconds," in *Proc. Int. Comput. Symp. (ICS)*, 2016, pp. 154–158.

[108] S. Attarha, K. H. Hosseiny, G. Mirjalily, and K. Mizanian, "A load balanced congestion aware routing mechanism for software defined networks," in *Proc. Iranian Conf. Electr. Eng. (ICEE)*, May 2017, pp. 2206–2210.

[109] R. Wazirali and R. Ahmad, "SDN-openflow topology discovery: An overview of performance issues," *Appl. Sci.*, vol. 11, no. 15, p. 6999, 2021.

[110] N. L. M. V. Adrichem, B. J. V. Asten, and F. A. Kuipers, "Fast recovery in software-defined networks," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, Sep. 2014, pp. 61–66, doi: 10.1109/EWSDN.2014.13.

[111] N. L. M. Van Adrichem, F. Iqbal, and F. A. Kuipers, "Computing backup forwarding rules in software-defined networks," 2016, *arXiv:1605.09350*.

[112] J. Kempf, E. Bellagamba, A. Kern, and A. Ab, "Scalable fault management for OpenFlow," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2012, pp. 6606–6610.

[113] D. Gyllstrom, N. Braga, and J. Kurose, "Recovery from link failures in a smart grid communication network using OpenFlow," in *Proc. IEEE Int. Conf. Smart Grid Commun. (SmartGridComm)*, Nov. 2014, pp. 254–259.

[114] 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 *Proc. 10th Int. Conf. Design Reliable Commun. Netw. (DRCN)*, Apr. 2014, pp. 1–8, doi: 10.1109/DRCN.2014.6816141.

[115] C. Cascone, L. Pollini, D. Sanvito, A. Capone, and B. Sans, "SPIDER: Fault resilient SDN pipeline with recovery delay guarantees," in *Proc. IEEE NetSoft Conf. Workshops (NetSoft)*, Jun. 2016, pp. 296–302.

[116] 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)*, Oct. 2016, pp. 95–99.

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

[118] M. Shojaee and M. Neves, "SafeGuard: Congestion and memory-aware failure recovery in SD-WAN," in *Proc. 16th Int. Conf. Netw. Service Manag. (CNSM)*, Nov. 2020, pp. 1–7.

[119] S. Sharma and D. Staessens, "Fast failure recovery for in-band OpenFlow networks," in *Proc. 9th Int. Conf. Design Reliable Commun. Netw.*, 2013, pp. 52–59.

[120] A. Capone, C. Cascone, A. Q. T. Nguyen, and B. Sanso, "Detour planning for fast and reliable failure recovery in SDN with OpenState," 2014, *arXiv:1411.7711*.

[121] Y. Yu, L. Xin, C. Shanzhi, and W. Yan, "A framework of using OpenFlow to handle transient link failure," in *Proc. Int. Conf. Transp., Mech., Electr. Eng. (TMEE)*, Dec. 2011, pp. 2050–2053, doi: 10.1109/TMEE.2011.6199619.

[122] 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, pp. 1066–1075, 2013, doi: 10.1364/JOCN.5.001066.

[123] B. Isyaku, M. Soperi, M. Zahid, and M. B. Kamat, "Software defined networking flow table management of OpenFlow switches performance and security challenges: A survey," *Future Internet*, vol. 12, no. 9, p. 147, 2020.

[124] S. Kim and S. Lumetta, "Evaluation of protection reconfiguration for multiple failures in WDM mesh networks," in *Opt. Fiber Commun. Conf., Tech. Dig.* Optica Publishing Group, 2003, pp. 1–4, 2003, Paper TuI7.

[125] 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.

[126] Z. Cheng, X. Zhang, Y. Li, S. Yu, R. Lin, and L. He, "Congestion-aware local reroute for fast failure recovery in software-defined networks," *J. Opt. Commun. Netw.*, vol. 9, no. 11, p. 934, Nov. 2017, doi: 10.1364/jocn.9.000934.

[127] C. Wang and H. Y. Youn, "Entry aggregation and early match using hidden Markov model of flow table in SDN," *Sensors*, vol. 19, no. 10, p. 2341, 2019.

[128] Z. Zhu, Q. Li, S. Xia, and M. Xu, "CAFFE: Congestion-aware fast failure recovery in software defined networks," in *Proc. 27th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2018, pp. 1–9, doi: 10.1109/ICCCN.2018.8487363.

[129] S. Hegde, S. G. Koolagudi, and S. Bhattacharya, "Path restoration in source routed software defined networks," in *Proc. 9th Int. Conf. Ubiquitous Future Netw. (ICUFN)*, Jul. 2017, pp. 720–725.

[130] A. L. I. Malik, B. Aziz, C. Ke, H. A. N. Liu, and M. O. Adda, "Virtual topology partitioning towards an efficient failure recovery of software defined networks," in *Proc. Int. Conf. Mach. Learn. Cybern. (ICMLC)*, 2017, pp. 646–651.

[131] H. Kim, J. R. Santos, Y. Turner, M. Schlansker, J. Tourrilhes, and N. Feamster, "CORONET: Fault tolerance for software defined networks," in *Proc. 20th IEEE Int. Conf. Netw. Protocols (ICNP)*, Oct. 2012, pp. 1–2.

[132] P. M. Mohan, T. Truong-Huu, and M. Gurusamy, "TCAM-aware local rerouting for fast and efficient failure recovery in software defined networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2015, pp. 1–6, doi: 10.1109/GLOCOM.2015.7417309.

[133] J. Chen, J. Chen, J. Ling, and W. Zhang, "Failure recovery using vlan-tag in SDN: High speed with low memory requirement," in *Proc. IEEE 35th Int. Perform. Comput. Commun. Conf. (IPCCC)*, Dec. 2016, pp. 1–9, doi: 10.1109/PCCC.2016.7820627.

[134] H. Liaoruo, S. Qingguo, and S. Wenjuan, "A source routing based link protection method for link failure in SDN," in *Proc. 2nd IEEE Int. Conf. Comput. Commun. (ICCC)*, Oct. 2016, pp. 2588–2594, doi: 10.1109/CompComm.2016.7925166.

[135] P. Thorat, R. Challa, S. M. Raza, D. S. Kim, and H. Choo, "Proactive failure recovery scheme for data traffic in software defined networks," in *Proc. IEEE NetSoft Conf. Workshops (NetSoft)*, Jun. 2016, pp. 219–225, doi: 10.1109/NETSOFT.2016.7502416.

[136] 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 *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–6, doi: 10.1109/ICC.2016.7510886.

[137] B. Stephens, A. L. Cox, and S. Rixner, "Scalable multi-failure fast failover via forwarding table compression," in *Proc. Symp. SDN Res.*, 2016, pp. 1–12.

[138] 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, Dec. 2017.

[139] P. Thorat, S. Jeon, S. M. Raza, and H. Choo, "Pre-provisioning of local protection for handling dual-failures in OpenFlow-based networks," in *Proc. 13th Int. Conf. Netw. Service Manag. (CNSM)*, 2017, pp. 1–6.

[140] S. Feng, Y. Wang, X. Zhong, J. Zong, X. Qiu, and S. Guo, "A ring-based single-link failure recovery approach in SDN data plane," in *Proc. IEEE/IFIP Netw. Oper. Manag. Symp.*, Apr. 2018, pp. 1–7, doi: 10.1109/NOMS.2018.8406152.

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

[142] S. Q. Zhang, Q. Zhang, A. Tizghadam, B. Park, H. Bannazadeh, R. Boutaba, and A. Leon-Garcia, "TCAM space-efficient routing in a software defined network," *Comput. Netw.*, vol. 125, pp. 26–40, Oct. 2017, doi: 10.1016/j.comnet.2017.06.020.

[143] H. Li, Q. Li, Y. Jiang, T. Zhang, and L. Wang, "A declarative failure recovery system in software defined networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–6, doi: 10.1109/ICC.2016.7510887.

[144] M. Shojaee, M. Neves, and I. Haque, "Congestion and memory-aware failure recovery in SD-WAN," in *Proc. 16th Int. Conf. Netw. Service Manag. (CNSM)*, Izmir, Turkey, pp 1–7.

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

[146] S. Zhang, Y. Wang, Q. He, J. Yu, and S. Guo, "Backup-resource based failure recovery approach in SDN data plane," in *Proc. 18th Asia–Pacific Netw. Oper. Manag. Symp. (APNOMS)*, Oct. 2016, pp. 1–6, doi: 10.1109/APNOMS.2016.7737211.

[147] W. Xin-gang, "A link performance-based failure recovery approach in SDN data plane," in *Proc. 3rd Int. Conf. Multimedia Image Process.*, 2018, pp. 46–51.

[148] I. Haque, "Revive: A reliable software defined data plane failure recovery scheme," in *Proc. 14th Int. Conf. Netw. Service Manag. (CNSM)*, 2018, pp. 268–274.

[149] T. Hu, P. Yi, J. Lan, Y. Hu, and P. Sun, "FTLink: Efficient and flexible link fault tolerance scheme for data plane in software-defined networking," *Future Gener. Comput. Syst.*, vol. 111, pp. 381–400, Oct. 2020, doi: 10.1016/j.future.2019.11.015.

[150] R. K. Das, F. H. Pohrmen, A. K. Maji, and G. Saha, "FT-SDN: A fault-tolerant distributed architecture for software defined network," *Wireless Pers. Commun.*, vol. 114, no. 2, pp. 1045–1066, Sep. 2020, doi: 10.1007/s11277-020-07407-x.

BABANGIDA ISYAKU received the B.Sc. degree in computer science and information system from Oxford Brookes University, in 2012, and the M.Sc. and Ph.D. degrees in computer science from Universiti Teknologi Malaysia (UTM), in 2017 and 2022, respectively. He works with Sule Lamido University, Kafin Hausa, Jigawa, Nigeria. He is currently a Researcher at Universiti Teknologi Malaysia under the Post-Doctoral Fellowship Scheme. His research interests include software defined networks, routing, failure recovery, and flowtable management. He was a recipient of the Best Paper Award at IEEE Symposium on Computer Applications and Industrial Electronics, in 2020, and the Best Postgraduate Student Award from the Faculty of Computing, UTM.

KAMALRULNIZAM BIN ABU BAKAR (Member, IEEE) received the B.Sc. degree in computer science from Universiti Teknologi Malaysia, Malaysia, in 1996, the M.Sc. degree in computer communications and networks from Leeds Metropolitan University, U.K., in 1998, and the Ph.D. degree in computer science from Aston University, U.K., in 2004. He is currently a Professor with the Department of Computer Science, Universiti Teknologi Malaysia, and a member of the Pervasive Computing Research Group. His research interests include mobile and wireless computing, *ad-hoc* and sensor networks, information security, and grid computing. He is involved in many research projects and also a referee of several scientific journals and conferences. He is a member of ACM, the Internet Society, and the International Association of Engineering.

FUAD A. GHALEB received the B.Sc. degree in computer engineering from the Faculty of Engineering, Sana'a University, Yemen, in 2003, and the M.Sc. and Ph.D. degrees in computer science (information security) from the Faculty of Engineering, School of Computing, Universiti Teknologi Malaysia (UTM), Johor, Malaysia, in 2014 and 2018, respectively. He is currently a Senior Lecturer with the Faculty of Engineering, School of Computing, UTM. His research interests include vehicular network security, cyber security, intrusion detection, data science, data mining, and artificial intelligence. He was a recipient of many awards and recognitions, such as the Postdoctoral Fellowship Award, the Best Postgraduate Student Award, the Excellence Awards, and the Best Presenter Award from the School of Computing, Faculty of Engineering, UTM, as well as the best paper awards from many international conferences.

ABDULAZIZ AL-NAHARI received the B.Sc. degree in information technology from Al-Balqa Applied University, in 2005, the M.Sc. degree in computer science from The University of Jordan, in 2009, and the Ph.D. degree in computer science from the School of Computing, Faculty of Engineering, Universiti Teknologi Malaysia (UTM), in 2018. He has been working at the Programming Unit, Sana'a Community College, Sana'a, since 2009. He has been working as a Senior Lecturer at the UNITAR Graduate School, UNITAR International University, Malaysia, since June 2021. His research interests include computer networks, routing protocols in *ad-hoc* networks, machine learning, and data analytics.