IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 20 September 2023, accepted 30 November 2023, date of publication 4 December 2023,
date of current version 18 December 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3339381

RESEARCH ARTICLE

An Effective, Efficient and Scalable Link Discovery (EESLD) Framework for Hybrid Multi-Controller SDN Networks

ISMAIL AL SALTI AND NING ZHANG
Department of Computer Science, The University of Manchester, M13 9PL Manchester, U.K.
Corresponding author: Ismail Al Salti (ismail.alsalti@postgrad.manchester.ac.uk)

This work was supported by The University of Manchester.

ABSTRACT The emergence of Software-Defined Networking (SDN) has revolutionised network management, offering improved flexibility, programmability, and scalability through the introduction of centralised controllers. Such a controller, e.g. an SDN controller, typically uses an OpenFlow discovery protocol to establish and maintain a global view of the topology in the underlying network. An accurate global view of the network topology is essential for effective routing, load balancing, and deployment of mobility-based applications. However, in a hybrid multi-controller SDN network, the OpenFlow discovery protocol introduces repetitive operations, degrading effectiveness, efficiency, and scalability. This paper addresses this issue by presenting a novel link discovery framework for establishing and maintaining topology information in a hybrid multi-controller SDN network. The framework, named EESLD, uses an event-driven approach utilising the Bidirectional Forwarding Detection (BFD) protocol to detect direct and indirect SDN links in intra-domain and inter-domain networks. Additionally, EESLD uses the sFlow protocol to discover and monitor legacy links and employs a distributed messaging system to maintain a consistent network view across controllers. The EESLD framework has been implemented and evaluated on a Mininet emulator with an RYU controller and sFlow server. Performance evaluation results show that the EESLD framework can discover direct SDN links 10.3 times faster than OFDPv2, and indirect SDN links 12.9 times faster than BDDP in a network with 85 switches. Evaluation results also indicate that the sFlow-based link discovery outperforms the OSPF-based link discovery in legacy link discovery and removal times. These results show that the EESLD framework is a more effective, efficient and scalable solution for dynamic and large-scale hybrid multi-controller SDN networks.

INDEX TERMS Software-defined networking (SDN), topology discovery, link discovery protocols, OpenFlow protocol, hybrid SDN, multi-controller SDN.

I. INTRODUCTION

In recent years, the fast evolution of mobility, cloud computing, virtualisation, and multi-tenant networks has presented increasing challenges in managing traditional networks. In response to these challenges, Software-Defined Networking (SDN) has emerged as a viable solution for simplifying network management tasks and offering flexibility in managing network resources. SDN separates the control and data planes of network devices, enabling centralised network management and configuration. SDN has been widely adopted in data centres, campus networks, and wide area networks, providing significant benefits such as improved network programmability, flexibility and scalability. Notably, it has been adopted in Google's private backbone network called B4 [1] to interconnect their global data centre networks.

The SDN conceptually centralises the control plane in an entity known as the SDN controller. The controller communicates with and controls the devices within the network

The associate editor coordinating the review of this manuscript and approving it for publication was Xujie Li.

infrastructure, functioning as the central managing entity. The controller maintains a complete view of the network topology by periodically gathering topology information from the switches in the network. This approach allows for a comprehensive understanding of the underlying network structure, encompassing hosts, switches, and the connections between switches. Keeping this global view accurate and up-to-date is essential to ensure optimal performance of various core services and applications provided in the network, such as routing, host migration tracking, load balancing, and topology-based slicing. An SDN controller uses a topology discovery mechanism to discover and maintain network topologies.

The Link Discovery Service (LDS), a core service run on an SDN controller, is vital to topology discovery. It detects and tracks the presence of SDN links between network devices. Currently, the OpenFlow Discovery Protocol (OFDP) [2] is the most widely used in the SDN controllers for link discovery. OFDP uses the conventional Link Layer Discovery Protocol (LLDP) but incorporates a specific packet structure and adjusted operations to align it with the SDN framework [3]. LLDP is designed to discover one-hop SDN links between OpenFlow switches but not indirect multi-hop SDN links. Therefore, Broadcast Domain Discovery Protocol (BDDP) [4] was proposed to discover indirect SDN links that span multiple hops and are separated by legacy links within the same broadcast domain. BDDP is supported in open-source SDN controllers such as Floodlight [5] and OpenDayLight (ODL) [6]. However, the link discovery protocols, such as LLDP and BDDP, have several limitations when discovering links within dynamic and large-scale hybrid multi-controller SDN networks.

The first limitation relates to the discovery of various links within the network. It includes the inability to identify unicast and broadcast legacy links, as well as inter-domain links that connect different SDN domains, each managed by a separate controller. If these links are not discovered, it limits SDN controllers from having a comprehensive view of the network. As a result, it is challenging for the SDN controller to effectively manage and optimise the network's routing decisions. The second limitation is related to the repetitive method of link discovery, where controllers periodically query the status of each link. This process involves sending LLDP/BDDP packets to all the active ports within the network. As the network expands, the number of messages exchanged for complete link discovery and maintenance significantly increases. This high volume of traffic can lead to the risk of network congestion and adds substantial overhead to the controller's workload. Also, in dynamic environments like multi-tenant cloud data centres, where network states change frequently, the periodic and relatively long interval of link discovery (e.g., every 15 seconds with floodlight controller) fails to provide an up-to-date view of the network topology [3]. As a result, application-level network services such as routing may operate using outdated network configurations until they are updated again. There is, therefore, a need to address the limitations of link discovery protocols to achieve a more effective, efficient and scalable solution for dynamic and large-scale hybrid multi-controller SDN networks.

Works that have focused on addressing link discovery limitations in dynamic and large-scale hybrid multi-controller SDN networks can be categorised into two groups: SDN link discovery solutions and legacy link discovery solutions. SDN link discovery solutions aim to overcome the limitations by enhancing the existing link discovery protocols in SDN environments. For instance, the authors in [7] have proposed a modification to the existing OFDP implementation called OFDPv2. Their goal was to minimise the controller's overhead by reducing the number of Packet_Out messages sent during the OFDP discovery process. Chang et al. [8] proposed an approach that leverages LLDP-capable switches to transfer LLDP frames amongst neighbouring switches and relay any topology change to the controller. This approach offers benefits such as reduced CPU utilisation and the required number of packets for discovery. In addition, Rojas et al. [9] proposed a Tree Exploration Discovery Protocol (TEDP) to reduce the overhead of LLDP-based discovery. TEDP sends a single probe frame to a randomly chosen switch, which floods the network and explores the entire network simultaneously. Another approach for SDN link discovery was proposed by Ochoa-Aday et al. [10]. This approach divides the discovery process into phases and distributes the discovery functions among network nodes hierarchically. Hussain et al. [11] proposed a novel frame format to discover inter-domain link information in multi-controller SDN networks. Legacy link discovery solutions, on the other hand, focus on addressing the inability of existing link discovery protocols, such as LLDP and BDDP, to discover legacy links within hybrid SDN networks. One solution is using the Link State Advertisement (LSA) in Open Shortest Path First (OSPF) routing protocol to discover and maintain information about legacy links in hybrid SDN networks [12]. The intermediate SDN switch intercepts LSA messages from directly connected legacy switches. It then forwards them as packet_in to the SDN controller to build and update the topology information. Other solutions involve intercepting ARP [13] and STP [14] packets to facilitate the discovery of legacy links in the broadcast domain of hybrid SDN networks. Kuliesius and Giedraitis [15] use SNMP requests and trap messages via the SNMP southbound plugin SNMP4SDN [16] to acquire topology and device state information from legacy network devices. More detailed discussions of related SDN and legacy link discovery solutions are given in section V.

However, these prior works have limitations. First, current link discovery solutions are specifically designed for either SDN or conventional/legacy networks. In a hybrid SDN network that integrates multi-controller SDN infrastructures, these solutions fail to afford an SDN controller a holistic understanding of the complete network topology. This limitation restricts the ability of the controllers to effectively manage and optimise the network. The second

open issue is how to reduce the time it takes to detect topological changes. This detection time directly impacts SDN controllers' responsiveness to changes in network topology, especially in hybrid SDN networks that include legacy switches. Any changes in the network, such as adding or removing devices, can lead to changes in network link configurations and, therefore, the overall network topology. To ensure effective network management and optimal performance, it is crucial for SDN controllers to promptly detect and adapt to these topological changes. Furthermore, it is crucial for the link discovery solution to be designed with scalability while addressing the mentioned issues. This means that the solution should be able to handle larger networks without sacrificing performance or introducing significant overhead to the SDN controllers. Link discoveries require regular operations, which can result in increased communication and computational overhead for both SDN controllers and switches. To minimise these overheads, it is crucial to design efficient link discovery operations as the network expands, the number of network switches and links increases, or when there are more dynamic changes in the network topology. Therefore, how to effectively discover and maintain a comprehensive view of dynamic and large-scale hybrid multi-controller SDN networks while minimising overloads as much as possible, is still an open research issue.

As part of our efforts to tackle this issue, this paper describes the design and evaluation of EESLD, a novel framework for effective, efficient and scalable link discovery in hybrid multi-controller SDN networks. The novelty of the EESLD framework lies in its ability to obtain and maintain a comprehensive view of the network topology in dynamic and large-scale hybrid multi-controller SDN networks. To achieve this, EESLD employs four novel methods. First, the Event-Driven SDN Link Discovery (EDSLD) method uses the Bidirectional Forwarding Detection (BFD) protocol to discover direct and indirect SDN links within intra-domain and inter-domain networks. Second, the Priority-Based Link Status Inspector (PILSI) method is used to monitor the SDN links status by selectively polling a few critical switches, which ensures coverage of all discovered SDN links. Additionally, it prioritises switch updates based on their respective importance. Third, the sFlow-Based Legacy Topology Mapping (FDLTM) method uses the sFlow protocol to discover and monitor broadcast and unicast legacy links within the network. Fourth, the Network-Wide Topology Consistency (NWTC) method employs a distributed publish-subscribe messaging system to synchronise and maintain a consistent view of the network status across SDN controllers. This study offers the following significant contributions:

- We propose an effective, efficient, and scalable link discovery framework called EESLD. EESLD provides a comprehensive view of the network topology in dynamic and large-scale hybrid multi-controller SDN networks. The EESLD framework incorporates several novel methods to address the limitations of the existing link discovery protocols, such as LLDP and BDDP.
- We implemented the EESLD framework on the Mininet emulator with the RYU controller and sFlow server. In addition, we evaluated the performance of EESLD over different network scales. The experimental results were compared with those of the state-of-the-art protocols.

The rest of the paper is organised as follows: Section II overviews OpenFlow-based Software Defined Networks (SDN), hybrid SDN, and the topology discovery mechanism. Section III introduces the use case and motivation for our research. Section IV presents the requirements specification. Section V comprehensively analyses the existing solutions and their limitations. Section VI introduces the high-level ideas used to design the EESLD framework. Section VII describes the assumptions about the network environment and system operation. Section VIII provides a detailed explanation of the EESLD architecture and its components. Section IX presents the performance evaluation of the EESLD framework. Section X presents the limitations and future works of the EESLD framework. Finally, the conclusions are drawn in section XI.

II. BACKGROUND

This section provides an overview of OpenFlow-based Software Defined Networks (SDN), hybrid SDN and topology discovery mechanisms in traditional and SDN networks.

A. SOFTWARE-DEFINED NETWORKING

Software-Defined Networking (SDN) is a programmable framework that decouples the control plane from the data plane, allowing one control plane to manage multiple devices. In traditional networks, the control logic is distributed across network devices, making it difficult to manage and configure large-scale networks. SDN addresses this issue by offering a centralised control plane that simplifies network management via programmable interfaces. This centralised control plane increases flexibility, scalability, and efficient network configuration, making it a preferred choice for modern network architectures.

As shown in Figure 1, the SDN architecture comprises three main components: the application, control, and infrastructure layer [17], [18], [19], [20]. Each layer is responsible for specific functions within the architecture. The application layer provides services to end-users, the control layer implements network policies and manages traffic flows, and the infrastructure layer consists of network devices. Moreover, the application and control layers communicate via an unstandardised northbound API. The Representational State Transfer (REST) protocol is the most commonly used northbound interface [21]. The OpenFlow protocol is widely used as the southbound API for communication between the SDN controller and infrastructure devices [22].

Although SDN presents numerous advantages, its full implementation in networks is often restricted by various

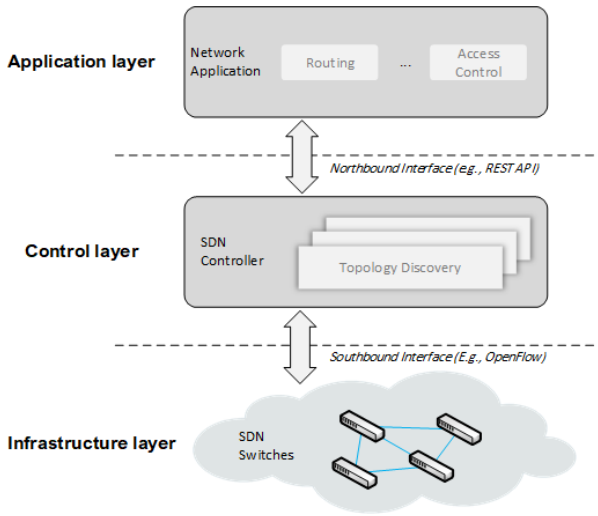


FIGURE 1. General architecture of SDN layers.

factors, including the complexity of the deployment, organisational constraints, and technical limitations. To address these challenges, a hybrid approach that integrates SDN with traditional networking could provide a more feasible and progressive path for existing networks to transition towards SDN.

B. HYBRID SDN

Hybrid SDN is a network architecture that merges centralised and decentralised approaches to configure, control, and manage network behaviour. Unlike traditional switches that rely on distributed algorithms such as Interior Gateway Protocol (IGP) for traffic routing management, an SDN controller routes traffic based on a global perspective. Combining these approaches gives rise to a hybrid SDN architecture where some traffic follows the conventional route while the SDN controller controls others. An example can be found in Google's B4 project [23], where they integrated an SDN application with a routing protocol to enable interaction between SDN switches and traditional routing protocols like OSPF.

The survey by Amin et al. [24] identifies two primary types of hybrid SDN deployments. These include integrating SDN switches into legacy networks and utilising hybrid switches supporting traditional networking functions and SDN capabilities. The gradual transition from traditional infrastructure to SDN is both cost-effective and technically feasible, as highlighted in the study by Hong et al. [12]. Therefore, hybrid SDN has become a favoured transitional solution for many organisations, merging the strengths of both traditional networking and SDN technology in a balanced compromise.

C. TOPOLOGY DISCOVERY SERVICE

Topology discovery is a critical component of network administration and operation. The significance of topology discovery is the ability to provide a comprehensive visualisation of the network's structure for troubleshooting, load balancing, identifying potential bottlenecks, and planning for future growth. This section describes the topology discovery process in both traditional and SDN networks.

1) TOPOLOGY DISCOVERY IN TRADITIONAL NETWORKS

Topology Discovery in the context of traditional networks forms the basics of effective routing protocols and switching techniques. The network typology discovery methods in traditional networks can be classified into physical and logical topology discovery [25]. The physical topology discovery is discovering the physical connectivity among entities in a network. The logical topology discovery, on the other hand, aims to establish logical connections based on the network components' IP addresses.

In detail, physical topology discoveries are usually based on MAC address tables collected from switches. The techniques used for this purpose include SNMP, ARP, and STP methods. The SNMP-based method queries every network device individually and then pieces the complete topology information together. The SNMP protocol enables communication between a manager and an agent on a managed device. The agent accesses the Management Information Base (MIB) database through initiated GET or SET requests from the manager. Moreover, agents can automatically send SNMP traps to notify managers of events such as topology changes. ARP-based method discovers network entities from the ARP table of any switch or router in the network. ARP maps an IP address to a physical (MAC) address on a local network. Gratuitous ARP (GARP) serves as a specialised form of ARP designed to keep devices on a local network informed about any changes in IP-to-MAC address mappings. By sending out a GARP frame, a network device announces its presence within the network, typically triggered by device boot-up or a change in the status of a specific link.

Moreover, the STP-based approach indirectly aids in discovering the structure of a network by identifying its active topology using information about existing links and their statuses. Conventional network nodes within a connected segment transmit STP messages through all available ports except those directly connected to user-end devices. Every STP message carries information, such as switch MAC address and port ID, that can be utilised to build network topology views.

Logical topology discoveries are used to construct a logical view of the network topology from layer-3 information. This can be achieved through the exchange of control messages, such as the Link State Advertisement (LSA) in Open Shortest Path First (OSPF) or BGP speaker announcements in Border Gateway Protocol (BGP). The LSA messages are sent by each node in the network under specific conditions to maintain an up-to-date and accurate view of the network topology across all nodes. Using LSAs, OSPF ensures that routers in the network have the latest information about the addresses

Header | Payload

Destination MAC	Source MAC	Ethernet Type	Chassis ID TLV	Port ID TLV	TTL TLV	Optional TLVs	End TLV
01:80:C2:00:00:0E	Outgoing Port MAC	0X88CC	Local switch ID	Sending Port ID	Time to Live	E.g., System Description	End Signal of LLDP

FIGURE 2. The format of LLDP packets.

and connectivity of all interfaces in the network. BGP speaker announcements, on the other hand, provide information about routes between autonomous systems. The announcements are only sent when there is a change in the routing table or during the initial exchange of routes. This exchange of route information helps BGP speakers update their routing tables and ensure they have the most up-to-date information about the network topology. Moreover, SNMP can query layer-3 devices to obtain detailed information, including data from the device's routing table, which lists all known network IP addresses along with the corresponding next hop for each network.

2) TOPOLOGY DISCOVERY IN SDN

Topology discovery is a process the SDN controller uses to learn about the three main network entities: hosts, network equipment (e.g., switches), and the interconnected links between the switches. The SDN controller discovers the actual location of the hosts within the network by utilising the Host Tracking Service (HTS) [26]. OpenFlow switches are discovered during the initial handshake process with the controller. The links between switches are discovered and tracked by a Link Discovery Service (LDS) [26]. LDS can dynamically discover network links by leveraging the OpenFlow Discovery Protocol (OFDP).

OFDP is considered a de facto protocol for link discovery in current mainstream SDN controllers [3], [27]. The OFDP adopts the layer 2 Link Layer Discovery Protocol (LLDP) with a few modifications to the protocol operation for compatibility with the SDN architecture. The SDN controller intercepts LLDP packets and extracts topology information to create an abstract network view. Figure 2 shows the format of the LLDP packet, which is divided into the header and payload. The header contains a destination address, source address, and Ethernet type. The payload of the LLDP packet consists of a different set of Type-length value (TLV) fields. Some controllers maintain a distinct set of TLVs. The Chassis ID, Port ID, and Time To Live (TTL) TLVs are used to store the switch data path ID (dpid), port number, and timestamp, respectively. The Optional TLVs store additional information not required for the topology discovery process.

Figure 3 illustrates the discovery process of the unidirectional link between the two OpenFlow switches (denoted by S1 and S2). The discovery process can be divided into four steps.

Step 1. The SDN controller, denoted as C0, initiates a request to obtain information regarding the active ports of switch S1. Afterwards, C0 encapsulates each LLDP packet within a Packet-Out message for every active port identified in S1. Finally, the controller sends these encapsulated packets to switch S1.

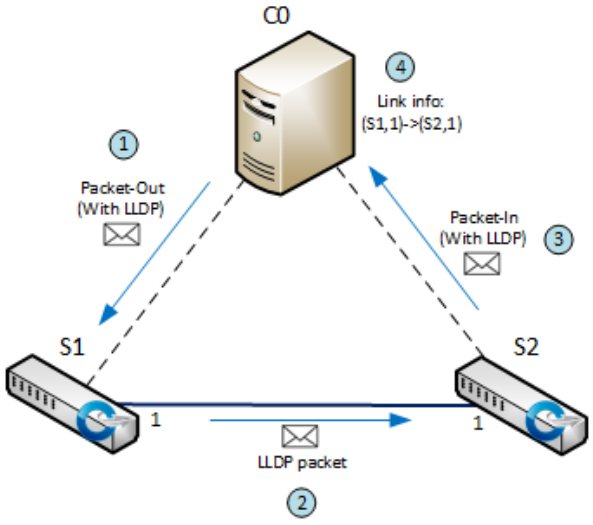


FIGURE 3. Discovering a unidirectional link from S1 to S2 using OFDP.

Step 2. Once the Packet-Out message reaches switch S1, it forwards the LLDP packet to port 1.

Step 3. When switch S2 receives the LLDP packet, it sends it to controller C0 as a Packet-In message with the LLDP packet encapsulated as payload.

Step 4. Controller C0 receives a Packet-In message with metadata for the destination switch's ID and port number. Using the LLDP payload and metadata, the LDS can identify a unidirectional link from switch S1 to switch S2.

Many SDN controllers can identify bidirectional links, which involve executing the same process in a reverse direction [28].

III. USE CASE AND MOTIVATION

We introduce a use case called Hybrid Multi-controller SDN Network (HMSN) as an example of a dynamic and large-scale data centre environment, as shown in Figure 4. The HMSN possesses several distinct characteristics:

- Multi-controller: this refers to the use of multiple controllers in the HMSN to manage and control the network. Multi-controller SDN distributes the control plane across multiple controllers, allowing for better scalability and fault tolerance. Each controller is responsible

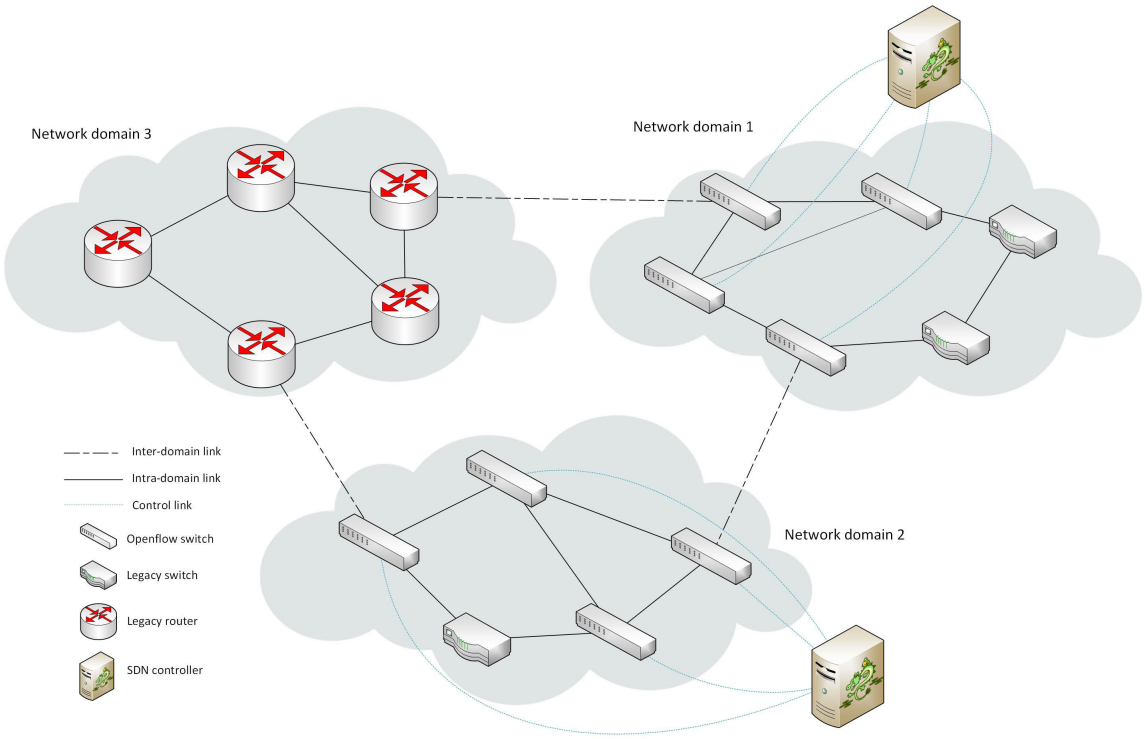


FIGURE 4. HMSN use case.

for a subset of the network, and they communicate with each other to ensure consistent network-wide policies. As network size increases, a single centralised controller may not be able to handle the increasing demand for flow processing. Therefore, multi-controller SDN is a promising solution for large-scale networks.

- Hybrid SDN Network: this involves the integration of traditional networking devices with modern SDN technology. This approach allows for a gradual transition to SDN while maintaining compatibility with legacy devices. The network is orchestrated through an SDN controller, which manages a portion of the devices while others continue to operate using conventional networking protocols. This combination protects existing network infrastructure and promotes future innovation.
- Dynamic network: this refers to a constantly evolving network that can have new switches or links added or removed at any moment. In addition, the changes to the topology occur frequently over a short period. The dynamic nature of these networks allows for adaptability to changing demands, potentially optimising performance by reducing latency and ensuring efficient resource allocation.

In such a use case, SDN controllers must be able to construct an immediate and up-to-date view of the network topology and status to act as centralised management systems and effectively address forwarding requests from network switches. The SDN controllers need to effectively discover and maintain the status of different types of links in a hybrid multi-controller SDN network, including inter and intra-domain SDN links, as well as unicast and broadcast legacy links. Moreover, to ensure optimal utilisation of network resources and enable network administrators to make well-informed decisions in real-time, SDN controllers need an efficient discovery process that promptly updates the central controller with minimal overhead. As the network grows, both in terms of nodes and complexity, the topology discovery mechanism must handle this growth without degradation in performance. Therefore, it would be desirable to have an effective, efficient and scalable network topology discovery solution to provide topology discoveries in a dynamic and large-scale hybrid multi-controller SDN environment.

IV. REQUIREMENT SPECIFICATIONS

Based on the HMSN use case, the following gives the requirements for designing an Effective, Efficient and Scalable Link Discovery (EESLD) framework.

A. FUNCTIONAL REQUIREMENTS

(FR1) Inter-domain and intra-domain SDN link discovery: it should be able to discover links between any pair of SDN switches within the same domain and between any pair of SDN switches in different domains, such as those managed by different SDN controllers.

(FR2) Indirect SDN link discovery: it should be able to discover indirect links between pairs of SDN switches separated by legacy switches.

(FR3) Legacy link discovery: it should be able to discover links between any pairs of legacy switches in both unicast and broadcast networks.

(FR4) Link status change detection: it should be able to detect link status changes for both direct and indirect links.

(FR5) Dynamic topology support: it should be adaptable to dynamic topological changes.

B. EFFICIENCY REQUIREMENTS

(ER1) Low bandwidth consumption: the bandwidth consumption introduced by the protocol should be as low as possible.

(ER2) Quick response to topology changes: the learning time required for an SDN controller to respond to any topology change should be as short as possible.

V. EXISTING SOLUTIONS AND ANALYSIS

A large body of research has focused on improving the efficiency of link discovery in pure SDN networks with a single controller. However, limited works have addressed link discovery's effectiveness, efficiency, and scalability in hybrid multi-controller SDN networks. A recent review study [29] discussed the state-of-the-art solutions that provided efficient and secured topology discovery solutions. In addition, several surveys [3], [30], [31] discussed an overview of the performance and scalability issues of the SDN-OpenFlow topology discovery. Since the scope of this paper is to investigate and evaluate the effectiveness, efficiency and scalability of SDN and legacy link discovery for hybrid multi-controller SDN networks, we categorise existing solutions into two main groups: SDN link discovery solutions and legacy link discovery solutions.

A. SDN LINK DISCOVERY SOLUTIONS

Depending on the approaches taken, existing solutions to SDN link discoveries can largely be classified into three groups: LLDP-based, tree exploration-based and hierarchical-based approaches.

1) LLDP-BASED APPROACH

The methods with this approach focus on enhancing the OpenFlow discovery protocol (OFDP) by exploiting the Link Layer Discovery Protocol (LLDP) capabilities. By using LLDP, researchers aim to reduce the overhead of discovery traffic, improve the scalability and performance of SDN, and make the discovery process more efficient.

Pakzad et al. [7] proposed a modification to the existing implementation of OFDP, known as OFDPv2. Their objective was to minimise the overhead of the OFDP discovery protocol by reducing the number of Packet_Out messages sent by the controller. Instead of individually sending an LLDP packet for each active port on every switch, they introduced a periodic transmission system where only one LLDP packet is sent per switch. The switches are then directed to duplicate this LLDP packet for each port and transmit it after modifying the Ethernet header's source MAC address - allowing neighbouring egress switches to identify the source port accurately. Consequently, this approach successfully reduces SDN controller-imposed overhead regarding Packet_Out messages.

Gebre-Amlak et al. [32] proposed a method in which multiple frequencies are used to represent various zones within the fixed tree network topology. The authors manually classified the tree topology into three distinct zones - core, aggregation, and edge zone. Since a failure in the core zone has a more significant impact on the network than a failure in the aggregation or edge zones, the core zone receives more frequent discovery messages. However, it is worth mentioning that this approach does not offer an automated means of determining network relevance and may not apply to other types of network topologies. Alenezi et al. [33] proposed using multiple discovery timers based on switch significance instead of a single timer for the entire network. This approach utilises centrality models to facilitate a method for identifying general significance. The proposed method is adaptable and can be applied to various network topologies, including tree, star, linear, grid, and wireless networks, without requiring manual configuration.

Additionally, Zhao et al. [34] introduced the Efficient and Secure Link Discovery (ESLD) scheme, which categorises the ports of an SDN switch into two types: 'Switch' and 'Host'. The switch ports establish connections between SDN switches, while the host ports connect to end users. ESLD optimises efficiency by restricting LLDP packet transmission to switch ports. Therefore, ESLD efficiency directly depends on the number of switch ports in the SDN networks. Another research work by Nehra et al. [35] proposed the Secure and Lightweight Link Discovery Protocol (SLDP), mainly developed to address various security threats. SLDP uses a modified packet format for link discovery, reducing unnecessary features from the standard LLDP frame. In addition, it classified ports into two categories: switch and host ports. SLDP packets are exclusively sent to switch ports. If the SDN controller receives an SLDP packet from the host ports, it will be immediately dismissed to avoid attacks.

Engineering et al. [36] proposed OFDPx, an enhanced version of OFDP that divides network links into multiple paths and utilises one probe packet per path for link discovery. This approach effectively reduces the probe packets required for complete link discovery. The implementation of OFDPx consists of the initial stage and the update stage. In the initial stage, the network topology is discovered following a procedure similar to standard OFDP. Subsequently, in the update stage, the discovered network links are divided into multiple paths, each detected using a single probe packet. The objective behind OFDPx is to minimise message exchange and optimise resource consumption.

Another research study by Gu et al. [37] called Im-OFDP aims to reduce the number of link discovery packets in topology discovery. The Im-OFDP topology discovery process is divided into initialising and updating stages. In the initialisation stage, the controller discovers network links and identifies supporting switches that cover all the discovered links using a minimum vertex cover algorithm. Additionally, predefined flow rules are installed on each supporting switch to direct LLDP packets to assigned ports. In the update stage, the controller sends a single LLDP packet to each supporting switch. Then, each switch encapsulates and distributes the LLDP packet to regulated ports. Instead of sending the LLDP directly to the controller as a Packet_In, the switch sends the LLDP packet back to the source switch. The source switch then forwards the packet to the controller as a Packet_In, establishing a bidirectional link between the switches.

The SDN controller periodically initiates the OFDP process to maintain an updated global topology view. However, this can lead to redundant and unnecessary discovery traffic being sent to the controller. To address this issue, researchers [8], [13], [38], and [39] have proposed event-driven discovery mechanisms that initiate network topology discovery only when changes occur. In their study, Azzouni et al. [38] propose a Secure and Efficient Topology Discovery Protocol called sOFTDP as a novel and efficient alternative protocol to the current OFDP. sOFTDP discovers the network topology based on specific events instead of periodically sending discovery packets. When it receives port-status messages, sOTFDP sends an LLDP packet to the corresponding port to learn newly added links. Additionally, it uses Bidirectional Forwarding Detection status messages to promptly detect link removal from the network topology.

Tarnaras et al. [13], [39] presented an event-based generic topology discovery algorithm. The authors used the Forwarding and Control Element Separation (ForCES) framework to construct the network topology by directly extracting LLDP messages from the switches. This method actively notifies the controller of any changes in the physical topology as they occur. Similarly, Chang et al. [8] proposed an approach that leverages LLDP-capable switches to transfer LLDP frames amongst neighbouring switches and relay any topology change to the controller.

The research stated that using a periodic operation method may generate excessive discovery traffic to the controller without topology changes, leading to scalability issues in large-scale networks. In addition, these works provide marginal or no reduction in link learning times. This may not be suitable for highly dynamic networks where links and devices undergo frequent changes, as it could cause delays in updating the topology information. On the other hand, the research that adopted the event-driven operation method for topology discovery often required switch modifications to support specific event-triggering functionalities. These modifications may not always be feasible to deploy due to hardware or software limitations, or they may not be cost-effective. Furthermore, the event-driven approach can lead to delays if the triggering events are not processed promptly, causing the network's topology information to be outdated or incorrect, which might affect application-level network services, such as routing.

2) TREE EXPLORATION-BASED APPROACH

Tree exploration link discovery offers an alternative approach to discovering SDN topology. It reduces traffic overhead by using a single probe frame that originates from the controller and floods the network. As this probe frame traverses the network, it gathers topology information at each hop and directly transmits it to the controller.

Rojas et al. [9] proposed a Tree Exploration Discovery Protocol (TEDP) to reduce the overhead of the point-to-point LLDP-based discovery process. TEDP achieves this by sending a single probe frame to a randomly chosen OpenFlow switch, which floods the network and simultaneously explores the entire network. Furthermore, TEDP can establish the shortest paths without introducing additional messages during discovery. While this method reduces the number of control plane packets for efficiency, the switches must be able to install their own forwarding rules, which requires modification of the switches for optimal performance.

Hussain et al. [40] propose a layer two link discovery scheme with a novel frame format to discover the network topology. The proposed scheme discovers the network topology by sending a singular probe frame to any OpenFlow switch. As the frame traverses through the network, each OpenFlow switch that receives it forwards it to the controller, subsequently updating the topology database. Hussain et al. [11] extend the scheme to discover inter-domain link information under a multi-controller environment's jurisdiction.

Jia et al. [41] proposed a new topology discovery protocol called Lightweight Automatic Discovery Protocol (LADP) for OpenFlow-based SDN networks. LADP enables the discovery of interconnected links between switches without requiring modifications to the OpenFlow protocol or switches. In LADP, the controller sends a single probe frame to a randomly selected root switch to initiate the exploration of the whole network. The root switch broadcasts the LADP frame to neighbouring switches, which then relay it to their neighbours while also forwarding it to the SDN controller. By analysing all received LADP frames, the SDN controller can obtain information about all interconnected links in the network.

Moreover, Alvarez-Horcajo et al. [42] proposed the Hybrid Domain Discovery Protocol (HDDP) to enhance network topology discovery in a wire SDN network that incorporates both SDN and non-SDN switches. HDDP employs a flooding-based network exploration model to discover non-SDN devices. Additionally, it introduces a lightweight agent on top of switches that implements HDDP and indirectly communicates topological information to the SDN controller. Subsequently, the authors in [43] enhanced

HDDP to facilitate topology discovery in various wireless networks.

In dynamic network environments, the tree exploration link discovery method may face difficulties adapting to frequent topology changes. This can lead to outdated or inaccurate information about the network structure and longer times for learning new links, as it requires traversing the entire network before detecting all modifications in topology. Furthermore, this approach necessitates switches installing memory-intensive rule groups or adjusting their forwarding rules.

3) HIERARCHICAL-BASED APPROACH

Hierarchical distributed discovery offers an alternative approach to exploring the topology of SDN networks. This approach divides the discovery process into phases and distributes the discovery functions among network nodes hierarchically. Ochoa-Aday et al. [10], [44] proposed an enhanced Topology Discovery Protocol (eTDP) that hierarchically distributes the discovery functions among network nodes. eTDP selects network nodes for aggregating the topology information and sends it to the controller. The selected nodes periodically send topology information to maintain an accurate global network view in the SDN controller. In detail, eTDP classified the SDN switches into three types (leaf, v-leaf and core). Leaf switches have only one adjacent switch, whilst those with more adjacent switches are called 'v-leafs'. The remaining SDN switches in the network are classified as the 'core'. Core switches aggregate topology information from neighbouring switches and send it to the SDN controller. The authors in [45] incorporate eTDP with failure recovery, which provides self-healing capabilities in SDN. Each forwarding element has to support the proposed algorithm through an agent. This follows a hybrid non-standard approach to SDN, potentially imposing additional workload on data plane devices. Furthermore, the aggregating of discovery topology information can lead to delay constraints. In networks that experience frequent changes in topologies, keeping up with these alterations can pose challenges, leading to outdated or inaccurate topology information.

B. LEGACY LINK DISCOVERY SOLUTIONS

In hybrid SDN networks, legacy link discovery solutions employ various approaches to provide the SDN controller with an accurate and current topology view of the legacy networks [46]. Based on these approaches, existing solutions can be classified into three main categories: Routing Protocols-based Approach, Link and Address Resolution-based Approach, and Network Management-based Approach.

1) ROUTING PROTOCOLS-BASED APPROACH

With this approach, network topology discovery is achieved through the exchange of control messages, such as the Link State Advertisement (LSA) in Open Shortest Path First (OSPF) or BGP speaker announcements in Border Gateway Protocol (BGP). These messages hold details about the network's link structure, which are then sent to the SDN controller. The controller extracts the information from these control messages to construct a complete view of the network topology and to detect topology changes.

The authors [12], [47], [48], [49], [50], [51], [52], [53], [54], and [55] use OSPF-LSAs messages that are flooded throughout the entire network to ensure all routers have a consistent and updated view of the network topology. These LSA messages are sent by each legacy switch under specific conditions, such as when a link goes up or down. The intermediate SDN switch intercepts LSA messages and forwards them as Packet_In to the SDN controller. Similarly, BGP speaker announcements provide information about paths to reach different parts of the network. These announcements are only sent when there is a change in the routing table or during the initial exchange of routes. Gämperli et al. [56] proposed a special BGP router, known as the cluster BGP speaker, which facilitates communication between external BGP routers and the SDN controller. Each cluster BGP speaker is connected to an SDN switch, which redirects BGP announcements to the SDN controller as Packet_In. The controller uses these announcements to build a map of the network topology.

A Routing Protocol-based approach offers advantages in discovering network topology and identifying nodes from various vendors. However, due to its slow convergence time, this approach does not adequately support changes to network topology. It can result in significant delays in network change detection by the SDN controller. In addition, it is applicable only for Layer 3 switches as it does not provide visibility into physical links and switches present at Layer 2.

2) LINK AND ADDRESS RESOLUTION-BASED APPROACH

This approach uses protocols like the Address Resolution Protocol (ARP) and Spanning Tree Protocol (STP) for network topology discovery in hybrid SDN networks. This approach discovers network topology by exchanging control messages at the link layer.

Tarnaras et al. [13] proposed using Gratuitous ARP (GARP), a specialised form of ARP, to discover the legacy links in a hybrid SDN environment. GARP is designed to keep devices on a local network informed about any changes in IP-to-MAC address mappings. By sending out a GARP frame, a network device announces its presence within the network, typically triggered by device boot-up or a change in the state of a specific link. The SDN controller's detection of these GARP frames has proven highly effective in discovering the network's topology.

The STP protocol is crucial in identifying links and handling link failures effectively. Within a connected

TABLE 1. Comparative table of the existing solutions of link discovery in hybrid SDN.(✓: satisfy the requirement, ✳:partially satisfy the requirement,✗: not satisfy the requirement).

The State-of-the-Art		References	F1	F2	F3	F4	F5	E1	E2
SDN link discovery	LLDP-based Approach	[7]	✳	✗	✗	✳	✗	✳	✗
		[39]	✳	✗	✗	✗	✗	✳	✳
		[13]	✳	✗	✳	✳	✗	✳	✗
		[32]	✳	✗	✗	✳	✳	✳	✳
		[34]	✳	✗	✗	✳	✗	✳	✗
		[35]	✳	✓	✗	✗	✗	✳	✗
		[38]	✳	✗	✗	✳	✓	✓	✓
		[37]	✳	✗	✗	✳	✗	✳	✗
		[36]	✳	✗	✗	✳	✗	✓	✗
		[8]	✳	✗	✗	✳	✗	✳	✗
		[33]	✳	✗	✗	✳	✳	✳	✳
	Tree Exploration-based Approach	[9]	✳	✗	✗	✳	✗	✳	✗
		[57]	✳	✓	✳	✳	✗	✳	✗
		[42]	✳	✗	✗	✳	✗	✳	✗
		[40]	✳	✗	✗	✳	✗	✳	✗
		[41]	✳	✗	✗	✳	✗	✳	✗
		[43]	✳	✗	✗	✓	✗	✳	✗
		[11]	✓	✓	✳	✳	✗	✳	✗
	Hierarchical-based Approach	[44]	✳	✗	✗	✳	✗	✳	✗
		[45]	✳	✗	✗	✳	✗	✳	✗
		[10]	✳	✗	✗	✳	✗	✳	✗
Legacy link discovery	Routing Protocols-based Approach	[47]	✗	✗	✳	✗	✗	✳	✗
		[48]	✗	✗	✳	✗	✗	✳	✗
		[56]	✗	✗	✳	✗	✗	✳	✗
		[50]	✗	✗	✳	✗	✗	✳	✗
		[51]	✗	✗	✳	✗	✗	✳	✗
		[12]	✳	✗	✳	✳	✗	✳	✗
		[52]	✗	✗	✳	✗	✗	✳	✗
		[53]	✗	✗	✳	✗	✗	✳	✗
		[54]	✗	✗	✳	✗	✗	✳	✗
		[55]	✗	✗	✳	✗	✗	✳	✗
		[49]	✗	✗	✳	✗	✗	✳	✗
	Link and Address Resolution-based Approach	[14]	✗	✗	✳	✗	✗	✳	✗
		[13]	✳	✗	✳	✳	✗	✳	✗
	Network Management-based Approach	[15]	✳	✗	✳	✳	✗	✳	✗
		[57]	✳	✓	✳	✳	✗	✳	✗

component, every legacy node sends out STP messages (BPDU) through its interfaces, excluding those connected to hosts. These BPDUs contain valuable information that the SDN controller can utilise to construct network topology views. In their study, Markovitch and Schmid [14] effectively use the Multiple STP (MSTP) protocol for automatic detection and localisation of network failures in the underlying physical network. Specifically, SDN switches of each network domain receive network updates via MSTP messages (BPDUs). These updates are then sent to the SDN controller, which uses them to identify the failed link and compute the impacted traffic.

The research works that use this approach are limited to the local network or broadcast domain, restricting its effectiveness in discovering network topology to a single network segment. Additional techniques or protocols may be necessary to address larger networks with multiple segments or subnets. Moreover, the constant exchange of BPDU messages can result in higher overheads and potential scalability issues. This can result in slower network topology updates and increased network congestion. In rapidly changing topologies, network convergence may not meet performance expectations.

3) NETWORK MANAGEMENT-BASED APPROACH

This approach employs the Simple Network Management Protocol (SNMP) to discover and build network topology in unicast and broadcast legacy networks in a hybrid SDN environment. This approach is especially suited for enterprise networks containing SNMP-capable devices like switches and routers. The SDN controller queries the SNMP agents on each device within the network to obtain topology information and create an accurate network view. Furthermore, the SDN controller utilises SNMP traps to receive proactive notifications from SNMP agents, allowing quick topology discovery and updates.

Kuliesius and Giedraitis [15] developed a HybN-Topo application to acquire topology and device state information from legacy network devices. This application communicates with the OpenDaylight (ODL) controller through the REST API to access its internal topology database. In order to collect data from legacy devices, the ODL controller uses SNMP requests and traps messages via the SNMP southbound plugin SNMP4SDN [16]. For successful registration with the controller, pre-configuration of legacy switches is required, including specific details such as an SNMP IP address and corresponding authentication credentials.

Hussain et al. [11] proposed a novel Indirect Controller to Legacy Forwarding (ICLF) scheme to discover SDN and legacy links in hybrid SDN networks. The ICLF initiates a single probe frame that traverses across the entire network. When a legacy switch receives this frame, it sends it from one switch to another until an SDN switch receives it. Upon receipt, the SDN switch forwards the frame to the controller, which uses the information to discover and update the network topology database. ICLF use SNMP requests and trap messages to gather information about the legacy switches and events related to the port status through the traps. Additionally, SNMP messages are used by ICLF for monitoring the status of legacy network links.

In complex network infrastructures with multiple devices, generating SNMP traps can overwhelm the SDN controller resources and make it difficult to identify critical events. SNMP traps are typically sent using User Datagram Protocol (UDP), which is connectionless and does not guarantee delivery. Consequently, lost traps may result in delays when identifying and responding to critical events. Moreover, traps often provide limited information about the event or issue that triggered them. Additional SNMP requests may be necessary to fully understand and resolve the issue, which can be time-consuming and resource-intensive.

Table 1 compares link discovery solutions in hybrid SDN networks based on the requirements described in Section IV. As the table shows, none of the existing solutions satisfy all requirements. The above-related work analysis indicates that there is room for improvements regarding the effectiveness, efficiency, and scalability of current SDN and legacy topology discovery approaches. Addressing these could enable more effective, efficient, and scalable link discovery in large-scale and dynamic network environments. This is due to the following observations.

- Firstly, existing link discovery solutions are either designed for SDN networks or traditional/ legacy networks. When applied to a hybrid network consisting of multiple controller SDN networks and legacy switches, these solutions do not enable an SDN controller to have a complete view of the topology in the entire network.
- The second open issue is how to shorten the time taken to detect any topological changes (i.e. topology change detection time) in the network, thus increasing SDN controllers' responsiveness to network topology changes in the presence of legacy switches. Adding or removing devices in the network causes changes in the network link configurations, thus network topology. Such changes should be detected promptly by the SDN controller. Any delay in detecting the changes may have a negative impact on the network management and performance. An efficient link discovery solution should allow SDN controllers to detect any topological changes in real time and to keep up-to-date topological data at any time.
- The third open issue is that while addressing the two issues mentioned above, the link discovery solution should also be designed to be scalable. Link discoveries require periodical operations. Each such operation imposes communication and computational overheads on SDN controllers and switches. Minimising these overheads when designing the link discovery operations as the network expands, the network switches increase, or the network topology changes more dynamically.

VI. HIGH-LEVEL IDEAS

This section presents the work on overcoming the limitations identified above by designing and evaluating an Effective, Efficient and Scalable Link Discovery (EESLD) framework that achieves link discoveries in an effective, efficient and scalable manner. By 'effective', we mean that the framework should be capable of discovering both direct and indirect SDN links within intra-domain and inter-domain networks. It should also be capable of discovering legacy links in unicast and broadcast traditional networks. By 'efficient', we mean the overhead introduced in network topology discoveries should be as low as possible. By scalable, we mean any measure that facilitates network topology discoveries should work equally well in both small and large, as well as static and dynamic networks. To this end, the following measures or high-level ideas have been taken in the design of this framework:

- We propose an approach combining three essential components to obtain a comprehensive view of network topology in a hybrid multi-controller SDN network. Firstly, we use an existing link failure detection protocol to efficiently discover direct and indirect SDN links within intra-domain and inter-domain networks. Secondly, we use a statistical sampling technique to identify legacy links within traditional unicast and broadcast networks. Finally, integrating a distributed database service allows for storing the global network topology and facilitates a publish-subscribe messaging pattern. This ensures a consistent and unified view of the entire network topology among SDN controllers.
- We propose an integrated approach combining event-driven and periodic topological discovery packet transmissions to enable adaptive and responsive link discovery in dynamic network environments with frequent topology changes. Specifically, the event-driven approach is used for discovering network links and detecting topology changes. In contrast, the periodic approach ensures the maintenance of up-to-date network status. This integrated strategy ensures a seamless and efficient response to evolving network conditions.
- To further reduce the overhead cost introduced by the periodic transmission link status packets and improve the scalability of link discovery in large-scale networks, we propose using a minimum number of switches responsible for providing the link status of the entire discovered SDN links in the networks. In addition, we group switches into two groups, each with a different importance level, L-Level and H-Level. The frequencies

of link status packet transmissions differ for the switches in different groups. The switches in the H-Level group send link status packets at a higher frequency. This way, the transmission and processing load imposed on the L-Level switches can be reduced.

VII. ASSUMPTIONS

To guide the design and evaluation of the EESLD framework, several assumptions about the network environment and system operation were made. The assumptions are as follows:

- (A1) Assume that there is a logically centralised controller, and the control and data planes are decoupled.
- (A2) Assumes an out-of-band control channel between the SDN controllers and the data planes, which use OpenFlow protocol as standard interfaces between the planes.
- (A3) Assume that each switch is preconfigured with its designated controller's IP address and TCP port number.
- (A4) Assume the underlying network is a hybrid SDN network comprising both OpenFlow and legacy switches.
- (A5) Assume the underlying network features direct SDN links between pairs of OpenFlow switches and indirect SDN links between pairs of OpenFlow switches separated by at least two legacy switches.
- (A6) Assume the underlying network includes traditional unicast and broadcast switches supporting sFlow technology for traffic monitoring, with sFlow agents preconfigured on these switches.

VIII. EESLD ARCHITECTURE

A. OVERVIEW

The EESLD framework aims to improve the effectiveness, efficiency, and scalability of the link discovery service in dynamic and large-scale hybrid multi-controller SDN networks. The EESLD framework addresses the limitations of existing link discovery protocols by employing four novel methods, as follows. The Event-Driven SDN Link Discovery (EDSLD) method uses the Bidirectional Forwarding Detection (BFD) protocol to identify direct and indirect SDN links in both intra-domain and inter-domain networks. Additionally, the Priority-Based Link Status Inspector (PILSI) method is utilised for selectively polling critical switches to monitor the status of SDN links, ensuring an up-to-date view of the SDN network. It also prioritises switch updates based on their respective importance. Furthermore, the sFlow-Based Legacy Topology Mapping (FDLTM) method leverages the sFlow protocol to discover and monitor broadcast and unicast legacy links within the network. Finally, employing a distributed publish-subscribe messaging system, the Network-Wide Topology Consistency (NWTC) method facilitates synchronisation across SDN controllers while maintaining a consistent view of the network status.

The EESLD architecture comprises four major modules, as shown in Figure 5. The modules are SDN Topology Discovery and Maintaining, SDN Topology Status Monitoring, Legacy Topology Discovery and Maintaining, and Global Topology Synchronisation. In the following sections, we explain each module and its submodule.

B. SDN TOPOLOGY DISCOVERY AND MAINTAINING MODULE

The SDN Topology Discovery and Maintaining (STDM) module plays a crucial role in discovering direct and indirect SDN links within the SDN network topology while also ensuring the real-time detection of any changes or modifications to the SDN topology. This module depends on the Bidirectional Forwarding Detection (BFD) protocol for discovering and monitoring SDN links. BFD protocol was adopted in SDN from a concept used in legacy technologies [58], and it is a simple hello protocol designed to detect link failure in a network [59]. The BFD protocol establishes a session on each link connecting pairs of routers or switches. By applying the BFD protocol, the STDM module effectively ensures accurate discovery and real-time monitoring of the SDN network topology. The STDM module has four submodules: active port BFD enabler, BFD packet handler, link completion, and Port status handler. The details of each submodule are discussed below.

The Active Port BFD Enabler (APBE) module activates the BFD protocol on the active ports of the switches discovered by the SDN controller. After the handshake between the switch and the controller using a three-way handshake (SYN, SYN/ACK, ACK), the controller sends a ''FEATURES_REQUEST'' message to the switch. The switch responds with a ''FEATURES_REPLY'' message that outlines its capabilities, such as flow statistics, table statistics, port statistics, group statistics, IP reassembly, queue statistics, etc. Then, the controller sends the ''OFPT_MULTIPART_REQUEST'' message with the ''OFPMP_PORT_DESC'' type to retrieve information about all the active ports associated with an OpenFlow switch. The switches reply with an ''OFPT_MULTIPART_REPLY'' message back to the controller. The message contains detailed information about the physical active ports of the switch. APBE module stores active port information in the Ports Information Database. Subsequently, the APBE module sends the BFD enable command for each active port on the switch using the OVSDB library in the RYU controller. To be more precise, it uses the ovs_vsctl.VSCtlCommand, which enables the BFD protocol in the active interfaces by manipulating the configuration database of Open vSwitch. The BFD session is configured with a minimum transmission interval equal to one millisecond and a minimum receive interval equal to a hundred milliseconds, as shown in Algorithm 1.

The BFD Packet Handler (BPH) module parses and extracts link information from incoming BFD packets. Once the BFD protocol has been activated on both ends of a specific link, the switch with the most recently enabled port will send

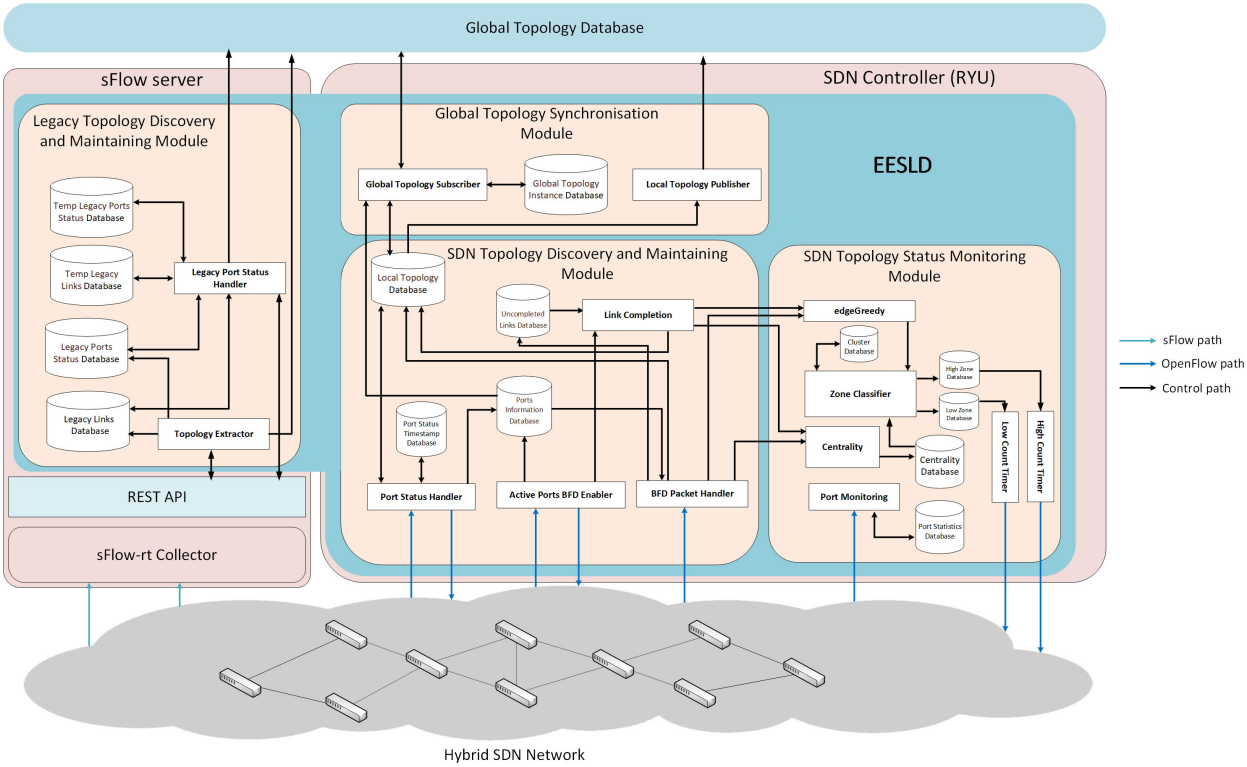


FIGURE 5. EESLD architecture.

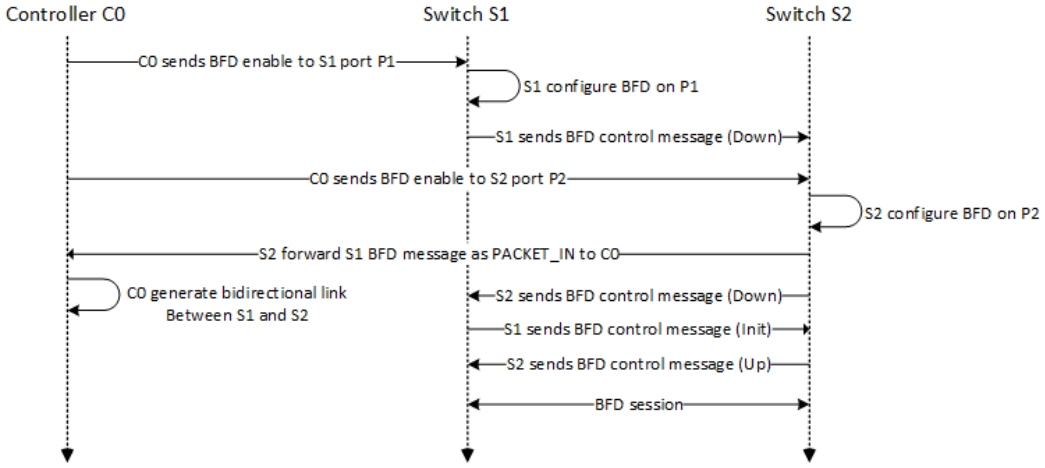


FIGURE 6. Link discovery sequence.

the BFD packet to the controller. Let us look at an example where we assume that we have an SDN controller (referred to as C0) connected to two OpenFlow switches, S1 and S2. These switches are interconnected, with port P1 on S1 linked to port P2 on S2. The controller C0, as illustrated in Figure 6, initiates a BFD session by first enabling it on port P1 of switch S1. Switch S1 initiates a BFD session with switch S2 by sending BFD control messages. Next, the controller enables BFD on port P2 of switch S2. Switch S2 starts sending BFD control messages to switch S1 in the process of creating a BFD session. During the session setup, S2 transmits BFD control messages to C0 as Packet_In messages. Typically, only one message is sent per session. The Controller C0 receives a Packet_In message that includes the metadata for the source Datapath ID and Port Number. Based on the BFD header and metadata, the BPH model discovers and constructs a link between switches S1 and S2, which is then stored in the Local Topology Database. The discovered link essentially consists of the Source Switch ID and Port Number, which are the origin of the BFD message, and the Destination

Algorithm 1 APBE Algorithm

```
Input: L (Multipart reply message (PORT_DESC))
Procedure: APBEnabler
Active_Ports_List ← L.ports
for each port in Active_Ports_List do
    Port_Name ← port.name.decode("utf-8")
    Run_command('set Interface', Port_Name, 'bfd=enable=true
bfd=min_tx=1 bfd=min_rx=100')
    Switch_Datapath ← L.datapath
    Switch_ID ← L.datapath.id
    Port_Number ← L.ports[port].port_no
    Port_MAC ← L.ports[port].hw_addr
    Add(Ports_Information_Database,[Switch_Datapath, Switch_ID,
Port_Number,Port_MAC, Port_Name])
Link_Completion_Algorithm(Switch_ID)
```

Algorithm 2 BPH Algorithm

```
Input: M ( Incoming BFD packet )
Procedure: Link discovery
Src_Switch_ID ← M.datapath.id
Src_Port_Number ← M.match['in_port']
Dst_Port_MAC ← M.ethernet.src
Eth_Dst_MAC_Address ← M.ethernet.dst
if Eth_Dst_MAC_Address ≠′ 00 : 23 : 20 : 00 : 00 : 01′ then
    return
else
    Dst_Switch_ID ← Get Ports_Information_Database
['Switch_ID'] where (Ports_Information_Database['Port_MAC']
== Dst_Port_MAC)
    Dst_Port_Number ← Get Ports_Information_Database
['Port_Number'] where (Ports_Information_Database['Port_MAC']
== Dst_Port_MAC)
    Dst_Port_Name ← Get Ports_Information_Database
['Port_Name'] where (Ports_Information_Database['Port_MAC']
== Dst_Port_MAC)
    Src_Port_MAC ← Get Ports_Information_Database
['Port_MAC'] where (Ports_Information_Database['Switch_ID']
== Src_Switch_ID) and (Ports_Information_Database['Port_Number']
== Src_Port_Number)
    Src_Port_Name ← Get Ports_Information_Database
['Port_Name'] where (Ports_Information_Database['Switch_ID']
== Src_Switch_ID) and (Ports_Information_Database['Port_Number']
== Src_Port_Number)
    Link_ID = (Src_Switch_ID,Src_Port_Number,Dst_Switch_ID,
Dst_Port_Number)
    if Src_Port_MAC is Null then
        Add(Uncompleted_Links_Database, [Src_Switch_ID,
Src_Port_Number, Src_Port_MAC=0, Src_Port_Name=0,
Dst_Switch_ID, Dst_Port_Number, Dst_Port_MAC,
Dst_Port_Name, Link_ID])
    else if Dst_Switch_ID is Null then
        Add(Local_Topology_Database, [Src_Switch_ID,
Src_Port_Number, Src_Port_MAC, Src_Port_Name, Dst_Switch_ID
='ExternalSwitch', Dst_Port_Number = 0, Dst_Port_MAC,
Dst_Port_Name = 'ExternalPort', Link_ID])
    else
        Add(Local_Topology_Database, [Src_Switch_ID,
Src_Port_Number, Src_Port_MAC, Src_Port_Name, Dst_Switch_ID,
Dst_Port_Number, Dst_Port_MAC, Dst_Port_Name, Link_ID])
        New_Link ← [Src_Switch_ID, Dst_Switch_ID]
        TopologyLinksList.append(New_Link)
        Vertex = EdgeGreedy(TopologyLinksList)
        Centrility(TopologyLinksList)
```

Switch ID and Port Number, which are identified from the received Packet_In message. Simultaneously with the link discovery, both switches implement a three-way handshake (Down, Init, Up) for the BFD session set-up. The BFD control messages are exchanged periodically to monitor the link status.

The working procedure of the BPH module is described in Algorithm 2. Apart from link information is derived directly from the incoming BFD packet. And other details are retrieved from the Ports Information Database. For instance, the Source Switch ID, Source Port Number and Destination Port MAC Address are extracted from the incoming BFD packet. The remaining information, including the Destination Switch ID, Destination Port Number, Destination Port Name, Source Port MAC Address, and Source Port Name, is obtained from the Ports Information Database. When the Source Port MAC Address is not found in the Ports Information Database, it typically signifies that the SDN controller has discovered the link before receiving the "OFPT_MULTIPART_REPLY" message. This situation, known as a race condition, is expected during network topology initialisation. In this case, the link is considered incomplete and subsequently stored in the Uncompleted Links Database. Another scenario where the Destination Switch ID is not found in the Ports Information Database. The discovered link is considered an inter-domain link between a pair of OpenFlow switches, each controlled by different SDN controllers. These links are stored in the Local Topology Database with the Destination Switch ID and Destination Port Name tagged as ExternalSwitch and ExternalPort, respectively. In addition, the Destination Port Number is assigned with the value zero. After each link discovery, the BPH model will trigger EdgeGreedy and Centrality submodules in the SDN Topology Status Monitoring module.

The Link Completion (LC) module fills in the missing information for links stored in the Uncompleted Links Database. As previously explained by the BPH module, these Uncompleted links are those with missing information that has yet to be filled in. The LC module is triggered by the APBE module when it receives a new list of the active ports of a specific switch, as shown in Algorithm 1. The functionality of the LC module is detailed in Algorithm 3. The LC module uses the Switch ID to identify any links within the Uncompleted Links Database that share the same Source Switch ID. It fetches the missing link details from the Ports Information Database if such links are found. Once all the link information has been completed, the LC module stores this link in the Local Topology Database. After that, it triggers the EdgeGreedy and Centrality submodules within the SDN Topology Status Monitoring module for further operation.

The Port Status Handler (PSH) module monitors topology changes in the SDN network by intercepting and extracting reasons for port modification from incoming port status messages. In addition, it enables/disables the BFD protocol from designated switch ports. The port status "OFPT_PORT_STATUS" message informs the controller of any port status changes on the network. This can be due to a port being added, modified, or removed. The message

Algorithm 3 LC Algorithm

```
Input: Switch_ID
Procedure: Link Completion
if Switch_ID in Uncompleted_Links_Database ['Src_Switch_ID'] then
    Links ← Get all links where Uncompleted_Links_Database ['Src_Switch_ID'] = Switch_ID
    for link in Links do
        Src_Switch_ID ← link['Src_Switch_ID'].value
        Src_Port_Number ← link['Src_Port_Number'].value
        Src_Port_MAC ← Get Ports_Information_Database ['Port_MAC'] where (Ports_Information_Database['Switch_ID'] = Src_Switch_ID) and (Ports_Information_Database['Port_Number'] = Src_Port_Number)
        Src_Port_Name ← Get Ports_Information_Database ['Port_Name'] where (Ports_Information_Database['Switch_ID'] = Src_Switch_ID) and (Ports_Information_Database['Port_Number'] = Src_Port_Number)
        Dst_Switch_ID ← link['Dst_Switch_ID'].value
        Dst_Port_Number ← link['Dst_Port_Number'].value
        Dst_Port_MAC ← link['Dst_Port_MAC'].value
        Dst_Port_Name ← link['Dst_Port_Name'].value
        Link_ID ← link['Link_ID'].value
        Add( Local_Topology_Database, [Src_Switch_ID, Src_Port_Number, Src_Port_MAC, Src_Port_Name,Dst_Switch_ID, Dst_Port_Number, Dst_Port_MAC,Dst_Port_Name, Link_ID])
        New_Link ← [ Src_Switch_ID, Dst_Switch_ID ]
        TopologyLinksList.append(New_Link)
        Vertex ← EdgeGreedy(TopologyLinksList)
        Centrility(TopologyLinksList)
```

Algorithm 4 PSH Algorithm

```
Input: M (Port status message)
Procedure: Topology maintenance
    Current_Time ← time.time()
if M.reason == Modify then
    Port_Name ← M.desc.name.decode("utf-8")
    if Port_Name not in Port_Status_Timestamp_Database ['Port'] then
        break
    Port_status ← Port_Status_Timestamp_Database['Status'] where Port_Status_Timestamp_Database['Port'] == Port_Name
    Timestamp ← Port_Status_Timestamp_Database['Timestamp'] where Port_Status_Timestamp_Database['Port'] == Port_Name
    if M.desc.state == 4 then
        if Port_status == 1 then
            Run_command('set Interface', Port_Name, 'bfd=enable=true bfd=min_tx=1 bfd=min_rx=100')
            Switch_Datapath ← M.datapath
            Switch_ID ← M.datapath.id
            Port_Number ← M.desc.port_no
            Port_MAC ← M.desc.hw_addr
            Add(Ports_Information_Database, [Switch_Datapath, Switch_ID, Port_number, Port_MAC,Port_Name])
        else if Port_status == 0 and (Current_Time - Timestamp) > 2 then
            Run_command('set Interface', Port_Name, 'bfd=enable=false')
            Time.sleep(0.02)
            Run_command('set Interface', Port_Name, 'bfd=enable=true bfd=min_tx=1 bfd=min_rx=100')
    else if M.desc.state == 0 then
        if Port_status == 4 and (Current_Time -Timestamp) > 2 then
            Port_MAC ← M.desc.hw_addr
            Remove (Local_Topology_Database, Link[ Dst_Port_MAC or Src_Port_MAC] == Port_MAC)
    else if M.desc.state == 1 then
        Port_MAC ← M.desc.hw_addr
        Remove (Local_Topology_Database, Link[Dst_Port_MAC or Src_Port_MAC] == Port_MAC)
    Add(Port_Status_Timestamp_Database, [Port=Port_Name, Status=M.desc.state, Timestamp=Current_Time])
```

contains information such as the reason for the change (ADD, DELETE, MODIFY), the port number, the status of the port (link down, blocked, live), and other configuration data related to the port. When a link is removed, the controller receives two distinct "OFPT_PORT_STATUS" messages. The initial message contains a port state descriptor value of 0, indicating that the respective switch port has undergone a modification. Subsequently, a second message is received where the port state descriptor value is set to 1, indicating that the link associated with that specific port has been brought down. When a link between two OpenFlow switches is established, the controller receives a single "OFPT_PORT_STATUS" message from each switch. This message carries a port state descriptor value set to 4, indicating that the port is active and the physical link is present. Based on our observations, whenever the BFD session is enabled or disabled on a specific link, the controller is notified via two messages from each corresponding switch; each message carries a port state descriptor value of 0 and 4, respectively.

The process of the PSH module is described in Algorithm 4. The PSH module retrieves the 'reason' field from port status messages. If the 'reason' value is 'Modify', it obtains the port's name from the message. Then, it uses the Port Name to acquire the port status and timestamp from the Port Status Timestamp Database. Furthermore, the module logs the time of message reception. The port state descriptor value of the message determines the following actions:

- **If the descriptor value is 4**, the PHS module check the port status. If the status is "1", the interface is functional. BFD is then enabled on the interface, and its information is stored in the Ports Information Database. If the status is "0" and two or more seconds have passed since the last timestamp, this signifies that the BFD session has terminated because the indirect SDN link port on the other end is disabled. Consequently, the BFD on the interface is disabled, and it waits for 20 milliseconds, then re-enables the BFD on the same interface.
- **If the descriptor value is 0**, the PHS module verifies if the port status is "4" and if more than 2 seconds have elapsed since the last timestamp. If both conditions are met, the BFD session has ended. This is due to the disconnection of the link between two legacy switches, where each legacy switch is connected to an OpenFlow switch, forming an indirect SDN link.
- **If the descriptor value is 1**, the PHS module interprets this as the interface being administratively turned off. As a result, it removes the corresponding link from the Local Topology Database.

Whenever the PHS module intercepts port status messages, it saves the Port Name, Status, and Timestamp of the switch port in the Port Status Timestamp Database.

C. SDN TOPOLOGY STATUS MONITORING MODULE

The SDN Topology Status Monitoring (STSM) module provides real-time updates on the status of the SDN links within the network topology. It achieves this by periodically polling a minimum number of switches that cover all the discovered SDN links in the network. The STSM module classifies the selected switches based on their importance in order to determine the frequency of polling link status updates from the switches. Therefore, it reduces the overhead introduced by the periodic transmission of the link status packets, and it improves the scalability of SDN link discovery in large-scale networks. The STSM module has six submodules: EdgeGreedy, Centrality, Zone Classifier, High Count Timer, Low Count Timer, and Port Monitoring. The following paragraphs provide detailed descriptions of each submodule.

The EdgeGreedy (EG) module selects the minimum number of switches covering all the discovered SDN links in the network. It uses the EdgeGreedy algorithm [60], a heuristic algorithm in network optimisation. This algorithm addresses the Minimum Vertex Cover problem, aiming to find the smallest set of switches (vertices) so that every network link (edge) connects to at least one of the selected switches. The algorithm uses a list of discovered SDN links, each comprising the Source Switch ID and Destination Switch ID. The EG module is triggered whenever a link is added to the Local Topology Database by either the BPH or the LC modules.

The EdgeGreedy Algorithm operates in two phases. In the first phase, the algorithm begins with an empty set called a cover. As it examines every link in the network graph when it encounters a link $\{vi, vj\}$ that is not yet included in the cover, it chooses the switch (vertex) from $\{vi, vj\}$ with the higher number of connections (degree). If both vi and vj have equal degrees, the algorithm selects the first switch. At the end of this phase, the algorithm has constructed a Vertex Cover (VC). In the second phase, the algorithm calculates a value called the ''loss value'' for each switch in the VC. This loss value refers to the potential impact on the VC if the vertex were to be removed. The algorithm then assesses each switch in the VC. If it encounters a switch v with a zero loss value, removing v will not affect the cover. Therefore, the algorithm removes v from the VC and recalculates the loss values for all the switches connected to v. The optimised VC represents the minimum number of switches necessary to cover all the links in the network.

The Centrality module measures switch significance for various network topologies using the Centrality models. In network theory, Centrality refers to measures that quantify the importance of a particular switch (or vertices) within a network [61]. There are various ways of calculating Centrality; common Centrality measures include Degree Centrality, Betweenness Centrality, Closeness Centrality, and Eigenvector Centrality. However, the Centrality module uses Degree and Betweenness Centrality to effectively assess each switch's significance. Degree Centrality is used to determine the direct influence of a switch based on the number of links or connections it has with other switches. The Degree Centrality of each switch is calculated by simply dividing the number of connections *degree* in each switch s by the total number of switches N in the network, as illustrated in Equation (1).

$$DegreeCentrality = \frac{degree(s)}{|N| - 1} \tag{1}$$

On the other hand, Betweenness Centrality measures the extent to which a switch acts as an intermediary for other pairs of switches. It is defined as the number of shortest paths between pairs of nodes that pass through the switch of interest. The formula represented by Equation (2) calculates how frequently a switch $u \in S$ has been used as a link along the shortest path between two other switches $s,t \in S$, where $\theta(s, t)$ denotes the total number of shortest paths between switches s and t, and $\theta(s, t|u)$ refers to the number of those paths passing through switch u.

$$BetweennessCentrality = \sum_{s,t,u \varepsilon S} \frac{\theta(s, t|u)}{\theta(s, t)} \tag{2}$$

Like the EG module, the Centrality module is initiated by either the BPH r or the LC modules whenever a new link is added to the Local Topology Database. The outputs of the Centrality module are stored in the Centrality Database for future use by the Zone Classifier module.

The Zone Classifier (ZC) module divides switches selected by the EG module into two groups based on the result of the Centrality module. Using a K-Means clustering algorithm with two clusters (k = 2), the module assigns each switch to a zone based on their importance, as shown in Algorithm 5. This importance value is determined by combining each switch's Degree Centrality and Betweenness Centrality values into a two-dimensional input for the K-Means algorithm. Switches of greater importance are assigned to the High Zone, while those of lesser importance are designated as part of the Low Zone. The zone designation is determined by comparing the total Centrality of the first switch in each cluster. The cluster with the highest total Centrality is considered the High Zone, while the other cluster is designated the Low Zone. The switch IDs for each zone are then stored in the High Zone Database and Low Zone Database, respectively.

The system uses two timing modules to collect port statistics from switches: the High Count Timer (HCT) and the Low Count Timer (LCT). These modules send an ''OFPT_MULTIPART_REQUEST'' message with a type of ''OFPMP_PORT_STATS'' to request data, such as the number of packets or bytes sent and received per port, packet drops and error counts. The HCT module focuses

Algorithm 5 ZC Algorithm

Input: *Vertex* (A list of selected switches)
Procedure: Zone classification
for switch **in** vertex **do**
    Switch_ID ← switch
    DC ← **Get** Cenerility_Database['Degree_centrality'] **where** (Ports_Information_Database['Switch_ID'] == *Switch_ID*)
    BC ← **Get** Cenerility_Database['Betweenness_centrality'] **where** (Ports_Information_Database['Switch_ID'] == *Switch_ID*)
    Add(Cluster_Database, [*Switch_ID, DC, BC*])
Classification_Data ← array(zip(Cluster_Database['DC'], Cluster_Database['BC']))
Cluster_Database['Cluster'] ← **Kmeans**(n_clusters=2).fit (Classification_Data)
First_Cluster ← **Get** Cluster_Database['Switch_ID'] **where** Cluster_Database['Cluster'] == 0
Second_Cluster ← **Get** Cluster_Database['Switch_ID'] **where** Cluster_Database['Cluster'] == 1
FC_Total_Centrility ← **sum**(Cenerility_Database['Degree _centrality'] == *First_Cluster*[0], Cenerility_Database ['Betweenness_centrality'] == *First_Cluster*[0])
SC_Total_Centrility ← **sum**(Cenerility_Database['Degree _centrality'] == *Second_Cluster[0]*, Cenerility _Database['Betweenness_centrality'] == *Second_Cluster[0]*)
if *FC_Total_Centrility* > *SC_Total_Centrility* **then**
    Add(High_Zone_Database, *First_Cluster*)
    Add(Low_Zone_Database, *Second_Cluster*)
else
    Add(High_Zone_Database, *Second_Cluster*)
    Add(Low_Zone_Database, *First_Cluster*)

Algorithm 6 PM Algorithm

Input: P (Port statistics reply)
Procedure: Monitors network ports statistics
Switch_ID ← P.datapath.id
for port **in** P.body **do**
    Port_ID ← (*Switch_ID*, port.port_no)
    if port.port_no > 1000 **then**
        return
    else if *Port_ID* **in** RemovedPortsList **then**
        return
    else if *Port_ID* **in** Port_Statistics_Database['Port_ID'] **then**
        Received_Packets ← Port_Statistics_Database['Received_Packets'] **where** Port_Statistics_Database['Port_ID'] == *Port_ID*
        Transmitted_Packets ← Port_Statistics_Database['Transmitted_Packets'] **where** Port_Statistics_Database['Port_ID'] == *Port_ID*
        if port.rx_packets == *Received_Packets* **then**
            LogWarning('WARNING: No data received in Port ID , *Port_ID* ')
            RemovedPortsList.**append**(*Port_ID*)
        else if port.tx_packets == *Transmitted_Packets* **then**
            LogWarning('WARNING: No data transmitted in Port ID *Port_ID*')
            RemovedPortsList.**append**(Port_ID)
        else
            Update(Port_Statistics_Database, [(Received_Packets, Transmitted_Packets) ∈ *Port_ID*])
    else
        Add(Port_Statistics_Database, [*Port_ID, Received_Packets, Transmitted_Packets*])

on switches in the High Zone Database, sending requests every five seconds. On the other hand, the LCT module sends requests every ten seconds for switches located in the Low Zone Database. This distinction in timing allows the system to prioritise the monitoring and gathering of statistics from switches in the High Zone, as they are considered to have a higher level of importance and may, therefore, require more frequent updates.

The Port Monitoring (PM) module intercepts incoming port statistics packets and extracts relevant information. When receiving a port statistics request from the HCT or LCT module, switches will respond with an ''OFPT_MULTIPART_REPLY'' message of type ''OFPMP_PORT_STATS'', including the active port statistics information such as the number of received and transmitted packets. The PM module then parses and extracts the packets received and transmitted on each port, as shown in Algorithm 6. Then, it compares these values with the previous records stored in the Port Statistics Database to detect whether the link has stopped sending or receiving packets. When the BFD session is established for the SDN link, the ports involved in the link periodically exchange BFD control messages. If one of the ports were to stop sending or receiving packets, it would indicate a potential link failure. Therefore, if the number of received or transmitted packets equals the previous record stored in the Port Statistics Database, a warning is sent indicating that the port corresponding to this link is currently down. Consequently, the link associated with this inactive port is removed from the Local Topology Database.

D. LEGACY TOPOLOGY DISCOVERY AND MAINTAINING MODULE

The Legacy Topology Discovery and Maintaining (LTDM) module is designed to discover and monitor legacy links in unicast and broadcast traditional networks while ensuring that topology updates are transmitted to the Global Topology Database. The LTDM module achieves this by leveraging the sFlow [62] sampling technology to extract valuable information from network traffic to build the network topology. Additionally, the LTDM module monitors network traffic and updates the Global Topology Database to maintain a current network view. The LTDM module has two submodules: the Topology Extractor and the Legacy Port Status Handler module. The following paragraphs provide detailed descriptions of each submodule.

The Topology Extractor (TE) module extracts and builds legacy network topology from the sFlow-RT [63] collector. In addition, it transmits network topology information to the Global Topology Database. The sFlow-RT collector is a software application running on a sFlow server, as shown in Figure 5. It collects sFlow datagrams from all the sFlow agents embedded in the legacy switches and routers in the network. The sFlow datagrams provide detailed information about network traffic and are periodically sent to the sFlow collector for analysis. The TE module uses REST API to retrieve the current network topology from the sFlow collector.

The process of retrieving and building the legacy network topology from the sFlow collector is shown in Algorithm 7. After requesting network topology information

Algorithm 7 TE Algorithm

```
Input: topology, ifname, ifadminstatus, ifoperstatus
Procedure: Legacy topology discovery
Response ← request.get('http://localhost:8008/topology/json')
Topology_Info ← Response.json()
Topology_Links ← Topology_Info['links']
Int_Value ← 0
String_Value ← 'L1-'
for key in Topology_Links do
    Link ← Topology_Links[key]
    Int_Value ← Int_Value + 1
    Unique_Key ← String_Value + Int_Value
    Redis.set(Unique_Key, Link)
    Node1 ← Link['Node1']
    Node2 ← Link['Node2']
    Port1 ← Link['Port1']
    Port2 ← Link['Port2']
    Add(Legacy_Links_Database,[Unique_Key, Node1,Node2,
Port1, Port2])
Ports_names ← request.get('http://localhost:8008/table/ALL/
ifname/json').json()
Ports_Admin_Status ← request.get('http://localhost:8008
/table/ALL/ifadminstatus/json').json()
Ports_Oper_Status ← request.get('http://localhost:8008/table/
ALL/ifoperstatus/json').json()
for port in len(Ports_names) do
    Port_ID ← Ports_names[port]
    Port_Admin_Status ← Ports_Admin_Status[port]
    Port_Oper_Status ← Ports_Oper_Status[port]
    Add(Legacy_Ports_Status_Database, [Port_ID,Port_Admin_Status
,Port_Oper_Status])
```

using "/topology/json" URL, the TE module receives a JSON representation of the current network topology. This representation is based on the sFlow datagrams assembled by the sFlow collector. The JSON object encompasses a list of links between sFlow-enabled switches/routers, encapsulating a current snapshot of the network's connectivity. Each legacy link is made up of four components: Node1 with its corresponding 'port1', and Node2 with 'port2'. Moreover, every legacy link is assigned a unique identifier. For example, "L1-1", where "L" stands for a legacy domain, and "1" signifies the first domain. The second "1" denotes the first discovered link within that domain. After that, the link details are stored in the Legacy Links Database and then transmitted to the Global Topology Database to be distributed to the SDN controllers. Additionally, the TE module retrieves all the connected ports' administrative and operational statuses from the sFlow collector. This information is then added to the Legacy Ports Status Database for future analysis by the Legacy Port Status Handler module.

The Legacy Port Status Handler (LPSH) module monitors the legacy network topology to detect any changes; it promptly updates the Global Topology Database to reflect these alterations. By regularly monitoring the status of ports in the legacy network, the LPSH module provides current and up-to-date information about the network topology to both Local and Global Topology Databases.

Algorithm 8 illustrates the operation of the LPSH module. It periodically requests the status of the link ports (administrative and operational) from the sFlow collector. This data is stored in a temporary database known as the Temp Legacy Ports Status Database. It then compares the data with the previous records in the Legacy Ports Status Database to identify any changes in the network topology. Once changes are detected, it promptly updates the Legacy Links Database and the Global Topology Database. For example, if one of the port's statuses is down, it will update both databases to indicate that the corresponding link is no longer functional. On the other hand, if one of the existing ports' statuses changes from down to up, or a new active port is discovered, it will query the topology links from the sFlow collector, and the links are then stored in the Temp Legacy Links Database. After that, it compares them with the links stored in the Legacy Links Database. The new links are added to the Legacy Links and Global Topology Database accordingly.

Algorithm 8 LPSH Algorithm

```
Input: topology, ifname, ifadminstatus, ifoperstatus
Procedure: Legacy topology maintenance
while True do
    Ports_names ← request.get('http://localhost:8008/table/ALL
/ifname/json').json()
    Ports_Admin_Status ← request.get('http://localhost:8008/
table/ALL/ifadminstatus/json').json()
    Ports_Oper_Status ← request.get('http://localhost:8008/
table/ALL/ifoperstatus/json').json()
    for port in len(Ports_names) do
        Port_ID ← Ports_names[port]
        Port_Admin_Status ← Ports_Admin_Status[port]
        Port_Oper_Status ← Ports_Oper_Status[port]
        Add(Temp_Legacy_Ports_Status_Database,[Port_ID,
Port_Admin_Status, Port_Oper_Status])
    Ports_update ← compare(Legacy_Ports_Status_Database,
Temp_Legacy_Ports_Status_Database)
    if not Ports_update.empty then
        if Ports_update.iloc[0]['Port_Admin_Status'] == 'down' or
Ports_update.iloc[0]['Port_Oper_Status'] == 'down' then
            Port_ID ← Ports_update.iloc[0]['Port_ID']
            Link_ID ← Legacy_Links_Database['Unique_Key']
where Legacy_Links_Database['Port1'] or Legacy_Links_Database[
'Port2'] == Port_ID
            Redis.delete(Link_ID)
            Remove(Legacy_Links_Database, [Link_ID])
        else if Ports_update.iloc[0]['Port_Admin_Status'] ==
'up' or Ports_update.iloc[0]['Port_Oper_Status'] == 'up' then
            Response ← request.get('http://localhost:8008/
topology/json')
            Topology_Info ← Response.json()
            Topology_Links ← Topology_Info['links']
            Int_Value ← 0
            String_Value ← 'L1-'
            for key in Topology_Links do
                Link ← Topology_Links[key]
                Int_Value ← Int_Value + 1
                Unique_Key ← String_Value + Int_Value
                Node1 ← Link['Node1']
                Node2 ← Link['Node2']
                Port1 ← Link['Port1']
                Port2 ← Link['Port2']
                Add(Temp_Legacy_Links_Database, [
Unique_Key, Node1, Node2, Port1, Port2])
            New_Update ← compare(Temp_Legacy_Links
_Database, Legacy_Links_Database).drop_duplicates()
            Add(Legacy_Links_Database, New_Update)
            Redis.set(New_Update)
    Clear(Temp_Legacy_Links_Database)
    Clear(Temp_Legacy_Ports_Status_Database)
```

E. GLOBAL TOPOLOGY SYNCHRONISATION MODULE

The Global Topology Synchronisation (GTS) module is critical for maintaining a consistent replica of the network-wide state among SDN controllers. It acts as a bridge between Local and Global Topology Databases and ensures that they are synchronised. The GTS module uses the publish-subscribe messaging pattern to disseminate network updates to all the relevant controllers. Each controller can share its local network state view with others, bypassing the delays often associated with the request-reply pattern of client-server communication. This enables rapid and synchronised updates to ensure that all the controllers maintain the most current information about the network's topology. The GTS module has two submodules: Local Topology Publisher and Global Topology Subscriber.

The Local Topology Publisher (LTP) module transmits local topology updates to the Global Topology Database. To ensure accuracy, any modifications made in the Local Topology Database automatically trigger the LTP module. Consequently, these updates are promptly reflected in the Global Topology Database and later disseminated to subscribed SDN controllers. Meanwhile, the Global Topology Subscriber (GTS) module receives topology updates from the Global Topology Database. By subscribing to specific communication channels, it actively retrieves update messages whenever changes occur in the network topology. These updates provide details about additions or removals of links within the network.

The GTS module starts by subscribing to one or multiple channels in the Redis database. This is achieved using the ''psubscribe'' method in Redis. Once subscribed, the GTS module enters a listening state. It actively listens for any messages being published to its subscribed channels. The GTS module receives the message when a topology information update is published to a subscribed channel. The module then processes this information, updating its local network topology to reflect these changes as shown in Algorithm 9.

When a ''set'' notification event is detected, this refers to the addition of a new link to the Global Topology Database. The GTS model queries the link's information from the Global Topology Database using the link ID. The module stores the link information in the Global Topology Instance Database for legacy links. It assigns it a ''G'' type, indicating a global link. For SDN links, the module first distinguishes whether it is an intra-domain or inter-domain link. Intra-domain links are directly stored in the Global Topology Instance Database. For inter-domain links, the module fills in any missing link information, such as Destination Switch ID, Destination Port Number, and Destination Port Name, obtained from the Ports Information Database. The links are then saved to the Local Topology Database. When a link is deleted from the Global Topology Database, a ''del'' notification event is triggered. The GTS model immediately removes the link from the Global Topology Instance Database using the link ID. The GTS module continually monitors its subscribed channels to ensure that it maintains an accurate and up-to-date view of the global network topology.

Algorithm 9 GTS Algorithm

Input: E (event message from Redis server)
Procedure: Global Topology Subscription
Event ← E['pattern'].decode('utf-8')
Key ← E['data'].decode('utf-8')
if *Event* == 'del' **then**
    Remove(Global_Topology_Instance_Database, link with key == *Key*)
else if *Event* == 'set' **then**
    New_link ← redis.**get**(*Key*).json()
    if *Key*[0] == 'L' **then**
        Link_Key ← *Key*
        Src_Switch_ID ← *New_link*['node1']
        Src_Port_Number ← *New_link*['port1'].split('h')[1]
        Src_Port_MAC ← 0
        Src_Port_Name ← *New_link*['port1']
        Dst_Switch_ID ← *New_link*['node2']
        Dst_Port_Number ← *New_link*['port2'].split('h')[1]
        Dst_Port_MAC ← 0
        Dst_Port_Name ← *New_link*['port2']
        Link_ID ← Src_Switch_ID + '-' + Src_Port_Number +'-' + Dst_Switch_ID + '-' + Dst_Port_Number
        Link_type ← 'G'
        Add(Global_Topology_Instance_Database, [Src_Switch_ID, Src_Port_Number, Src_Port_MAC, Src_Port_Name, Dst_Switch_ID, Dst_Port_Number, Dst_Port_MAC,Dst_Port_Name, Link_ID, Link_type])
    else if *Key*[0] == 'C1' **then**
        if *New_link*['Dst_Switch_ID'] == 0 **then**
            return
        else
            Dst_Port_MAC ← *New_link*['Dst_Port_MAC']
            Dst_Switch_ID ← **Get** Local_Topology '_Database[Switch_ID'] **where** (Local_Topology_Database['Port_MAC'] == *Dst_Port_MAC*)
            if *Dst_Switch_ID* == 0 **then**
                Remove(Local_Topology_Database, Link[Dst_Port_MAC] == Dst_Port_MAC)
                Add(Local_Topology_Database, [*Src_Switch_ID, Src_Port_Number, Src_Port_MAC, Src_Port_Name, Dst_Switch_ID, Dst_Port_Number, Dst_Port_MAC, Dst_Port_Name, Link_ID* ∈*New_link*])
    else if *thenKey*[0] == 'C2'
        if *thenNew_link*['Dst_Switch_ID'] == 0
            Dst_Port_MAC ← *New_link*['Dst_Port_MAC']
            Dst_Switch_ID ← **Get** Ports_Information _Database['Switch_ID'] **where** (Ports_Information_Database['Port_MAC'] == Dst_Port_MAC)
            Dst_Port_Number ← **Get** Ports_Information_Database ['Port_Number'] **where** (Ports_Information_Database ['Port_MAC'] == Dst_Port_MAC)
            Dst_Port_Name ← **Get** Ports_Information _Database['Port_Name'] **where** (Ports_Information_Database['Port_MAC'] == Dst_Port_MAC)
            Src_Switch_ID ← *New_link*['Src_Switch_ID']
            Src_Port_Number ← *New_link*['Src_Port_Number']
            Src_Port_MAC ← *New_link*['Src_Port_MAC']
            Src_Port_Name ← *New_link*['Src_Port_Name']
            Link_ID ← Src_Switch_ID + '-' + Src_Port_Number + '-' + Dst_Switch_ID + '-' + Dst_Port_Number
            Add(Local_Topology_Database, [*Src_Switch_ID,Src_Port_Number, Src_Port_MAC, Src_Port_Name, Dst_Switch_ID, Dst_Port_Number, Dst_Port_MAC,Dst_Port_Name, Link_ID])
            Redis.**set**(*Key*, Local_Topology_Database[-1])
            Link_type ← 'G'
            Add(Global_Topology_Instance_Database, [{*Src_Switch_ID, Src_Port_Number, Src_Port_MAC, Src_Port _Name, Dst_Switch_ID, Dst_Port_Number, Dst_Port_MAC, Dst_Port_Name, Link_ID*}∈*New_link*, *Link_type*])

TABLE 2. Experimental environment for EESLD.

Resource	Configuration
Test-bed	Mininet (emulation) version 2.3.0
Operating system	Ubuntu 18.04.5 LTS
Controller	RYU version 4.34
OpenFlow\Legacy switch	Open vSwitch version 2.16.0
Network packet analyser	Wireshark version 3.6.1
Distributed Database	Redis version 6.0.16

IX. PERFORMANCE EVALUATION

In this section, we evaluate the performance of the EESLD framework. This evaluation aims to assess the efficiency and scalability of the framework. The evaluation process involves several key metrics. Section IX-A provides details of the experimental configurations used in this study. Subsequently, the network topology discovery and maintenance performance are presented in Section IX-B. We then analyse the overheads imposed on the controller and switches in Section IX-C.

A. EXPERIMENTAL CONFIGURATIONS

The EESLD framework was evaluated using a simulation environment. The emulated testbed used Mininet [64] to simulate a realistic network with OpenFlow and legacy switches. We used a VMware Workstation to create three Virtual Machines (VMs) connected to the Local Area Network (LAN). The first VM ran an SDN controller called RYU [65]. Unlike other SDN controllers, such as Floodlight and OpenDaylight, RYU has been developed as an open-source and well-documented controller. The second VM ran the sFlow server, which acted as a sFlow-rt collector [63] with the event engine that processed the incoming sFlow datagrams. The third VM ran an open-source distributed database service (Redis) [66] for topological information distribution. Additionally, we used Wireshark [67] to analyse and capture the network traffic. Table 2 lists the details of the experimental environment for the evaluation of the EESLD framework's performance.

B. TOPOLOGY DISCOVERY AND MAINTENANCE

This section evaluates the EESLD framework's performance, particularly its ability to discover and maintain the network topology after the initial network structure has been established on the SDN controller. The key performance metrics examined included the times needed to: (1) discover direct and indirect SDN links; (2) detect direct and indirect link removals; (3) discover legacy links; and (4) detect the removal of legacy links. These measurements are essential for evaluating how efficiently and promptly the EESLD framework can adapt to changes in network topology.

1) SDN LINK DISCOVERY AND REMOVAL

To evaluate the performance of the EESLD framework in terms of SDN link discovery and removal, four network topologies were used, as summarised in Table 3. Each topology consisted of OpenFlow switches and legacy switches.

TABLE 3. Topologies and key parameters.

Topology	Switches	Active Ports	Links Between Switches	Hosts
Linear	8	22	7	8
Linear	20	58	19	20
FatTree	30	214	83	48
Tree,4,4	85	424	84	256

All of these switches were implemented using Open vSwitch (OVS) [68]. The legacy switches were configured to operate in standalone mode and were not connected to the controller. For all the experiments, the controller and switches were connected in out-band mode. Each experiment was repeated 20 times, and the results presented are the average values from these iterations.

The SDN link discovery time is how long it takes the SDN controller to discover an SDN link in the network topology. The time delay is measured by calculating the time elapsed between enabling link ports and adding the link to the topology database. Figure 7 illustrates a comparative view of the average direct SDN link discovery time for three methods, namely OFDP, OFDPv2, and EESLD, over four network scales: 8, 20, 30, and 85 switches. The chart shows that the EESLD method significantly outperforms both OFDP and OFDPv2 in terms of network discovery time across all network scales. For instance, in a network with 8 switches, the EESLD method had an average discovery time of 76.1 ms, approximately 5.4 times faster than OFDPv2 (412.7 ms) and around 5.7 times faster than OFDP (437.1 ms). In a network of 85 switches, the average discovery time for EESLD was 284.8 ms, which is approximately 10.3 times faster than OFDPv2 (2927.9 ms) and almost 31.9 times faster than OFDP (9088 ms). The EESLD framework eliminates the need to generate discovery packets by generating them locally on the switch as soon as the link ports are activated. This avoids the latency associated with constructing and transmitting discovery packets to the network, as required by OFDP and OFDPv2.

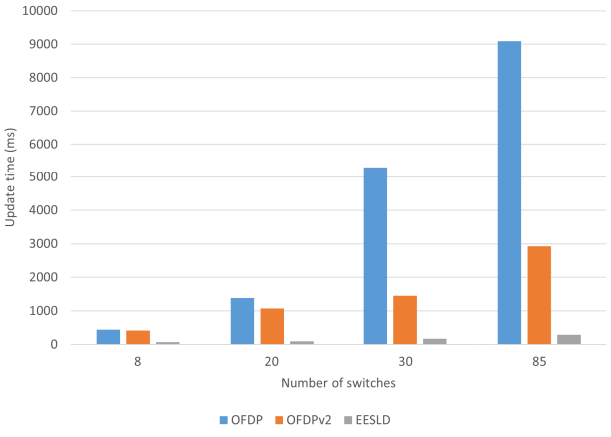


FIGURE 7. The average discovery time for direct SDN link.

Additionally, we evaluated the performance of EESLD against the Broadcast Domain Discovery Protocol (BDDP)

for discovering indirect SDN links. BDDP is a method used to discover multi-hop SDN links in hybrid SDN networks [4]. It is commonly supported in open-source SDN controllers such as Floodlight [5] and OpenDayLight (ODL) [6]. We implemented and evaluated the BDDP method in the Floodlight controller using two legacy switches to connect a pair of OpenFlow switches in the topology. This set-up created an indirect SDN link for our performance comparison.

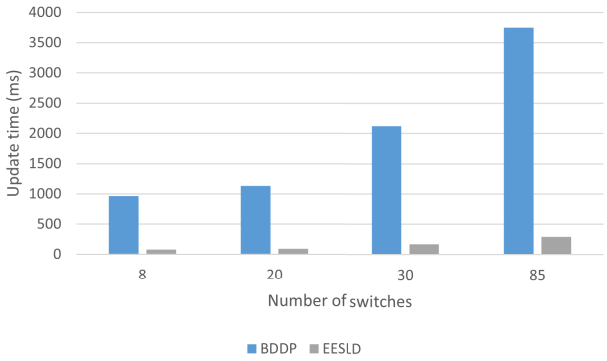


FIGURE 8. The average discovery time for indirect SDN link.

Figure 8 presents a comparative view of the average indirect SDN link discovery times for BDDP and EESLD over four network scales: 8, 20, 30, and 85 switches. The chart shows that EESLD was faster than BDDP across all network sizes. For a small network with 8 switches, EESLD was significantly faster than BDDP at 77.7 ms and 961.6 ms, respectively. This trend continues as the network size increases. At 20 switches, EESLD's discovery time was 95.4 ms compared to BDDP's 1134.2 ms. For larger networks of 30 and 85 switches, EESLD proved its efficiency with discovery times of 172.3ms and 290.2ms, respectively, compared to BDDP's times of 2122.2 ms and 3749.9 ms. The EESLD outperforms BDDP for discovering indirect SDN links. The reason for this is that BFD packets have a lightweight and optimised design for rapid link status detection. Moreover, BFD packets are generated locally, eliminating the delay caused by transmitting packets from the controller to the switches.

The detection time for a failed link refers to the time it takes for the SDN controller to detect and respond to the removal of a link in the network topology. The detection time is measured from when the link is removed to the point at which the controller detects the failure and updates the network topology accordingly. We measured the average detection time for direct and indirect SDN link failures.

Figure 9 displays the average detection time for direct link failures across four network scales: 8, 20, 30, and 85 switches. All methods demonstrated almost equivalent latency since link failures were detected by PORT DOWN events generated from the switches. However, the EESLD exhibited slightly faster detection times on larger network scales due to the reduced controller overhead. Unlike OFDP, which requires the controller to periodically send two discovery packets for each discovered link, OFDPv2 sends just one discovery packet per switch. The EESLD method allows for more immediate processing of other packets, including those for port status messages.

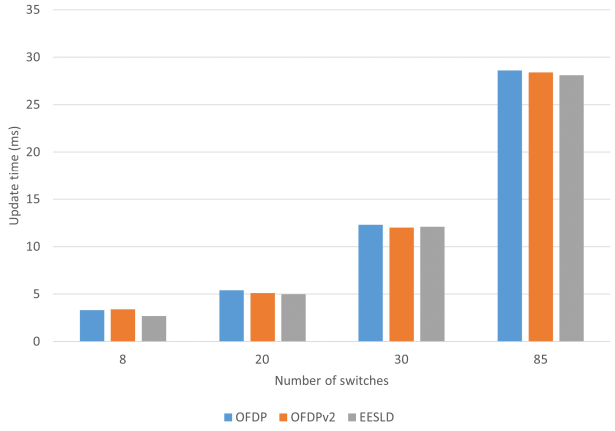


FIGURE 9. The average detection time for failed direct SDN link.

Additionally, during our analysis of the detection time for indirect link failures shown in Figure 10, we observed that the Floodlight controller could not detect indirect SDN link failures. The detection time is measured from the occurrence of a legacy link failure between the two legacy switches connecting the OpenFlow switches to the moment at which the SDN controller acknowledges it. The average detection time for indirect link failures tends to be longer than for direct link failures for two reasons. Firstly, the link discovery packet traverses more than one hop to reach the destination switch. Secondly, the BFD minimum receive interval is set to 100 milliseconds, which means that if the BFD session does not receive any packets within this time frame, it will be considered a link failure, and the switch notifies the SDN controller.

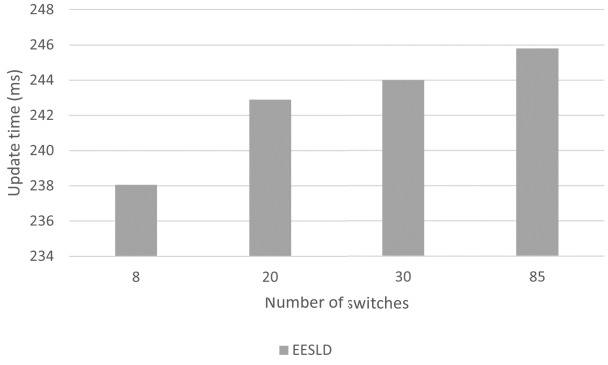


FIGURE 10. The average detection time for failed indirect SDN link.

Overall, the results for SDN link discovery and removal indicate the superior efficiency of the EESLD in terms of discovery time for direct and indirect SDN links, regardless of the network scale. This efficiency gain becomes more

pronounced as the network size increases, making EESLD suitable for dynamic and large-scale network environments.

2) LEGACY LINK DISCOVERY AND REMOVAL

We evaluated the average discovery and removal time for legacy links in the hybrid SDN network. All of the legacy switches operated in standalone mode and were pre-configured with sFlow agents to monitor and send sFlow packets to the sFlow collector. We evaluated three different intervals (1, 5, and 10 seconds) for sending sFlow packets to the sFlow collector. These results were compared with the most common existing approach to legacy link discovery: using OSPF-based link state advertisements for discovering and monitoring the legacy links.

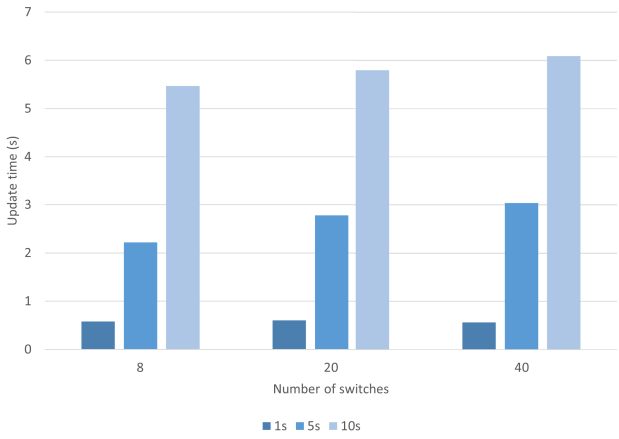


FIGURE 11. The average discovery time for new legacy link.

Figure 11 shows the average discovery time for legacy links using three legacy network topologies of 8, 20, and 40 switches. The 1-second interval achieves around 0.6 seconds for all network scales. However, for 5 and 10-second intervals, there is a slight increase in the average discovery time as the network scale increases. For example, with the 5-second interval, the average discovery time varies from 2.2 s for 8 switches to 3 s for 40 switches, while, with the 10-second interval, the average discovery time varies from 5.4 s for 8 switches to 6 s for 40 switches. Reducing the interval time for sending sFlow packets can lead to faster link discovery times, but, it may also increase the system's resource usage and network load.

On the other hand, Figure 12 shows the average removal time for legacy links. We can see that the average removal time for all network scales with a 1-second interval is around 0.6 s. This is equivalent to the average discovery time for a new link. Nevertheless, with longer intervals of 5 and 10 seconds, the average removal time increases slightly as the network scale increases. For example, with a 5-second interval, the average removal time varies from 1.9 s for 8 switches to 2.8 s for 40 switches. Also, with a 10-second interval, the average removal time varies from 5.1 s for 8 switches to 5.3 s for 40 switches. This indicates that the discovery of legacy links takes slightly longer

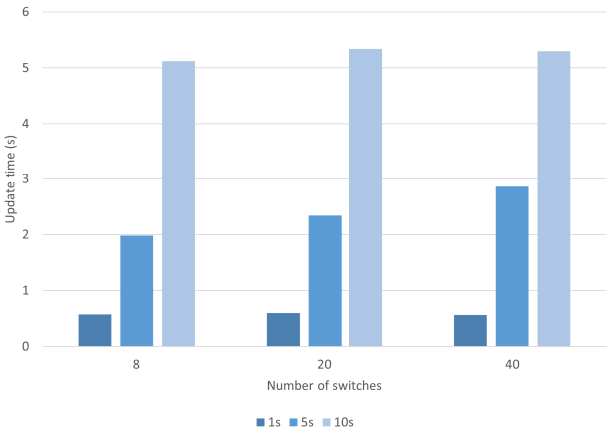


FIGURE 12. Average detection time for EESLD framework legacy link removal.

than the removal process, but the difference in time is minimal.

We compared legacy links' discovery and removal times using OSPF. This approach achieved network topology discovery by exchanging the link state advertisement messages. When the OpenFlow switch receives these advertisements, it forwards them to the SDN controller, which updates its internal database and maintains an accurate network topology view. We used hop counts of 2, 9, and 18 to represent the number of intermediate devices a packet must traverse to reach the closest OpenFlow switch. The network topology was simulated using the Graphical Network Simulator (GNS3) [69].

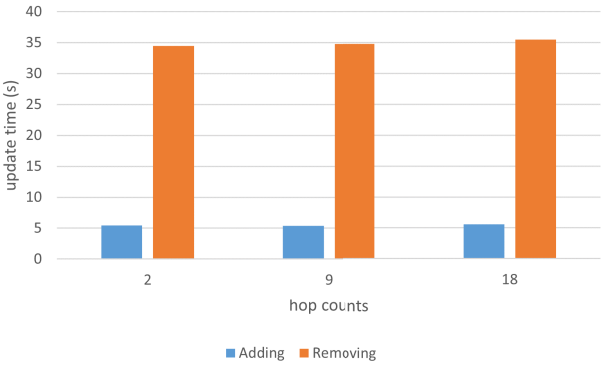
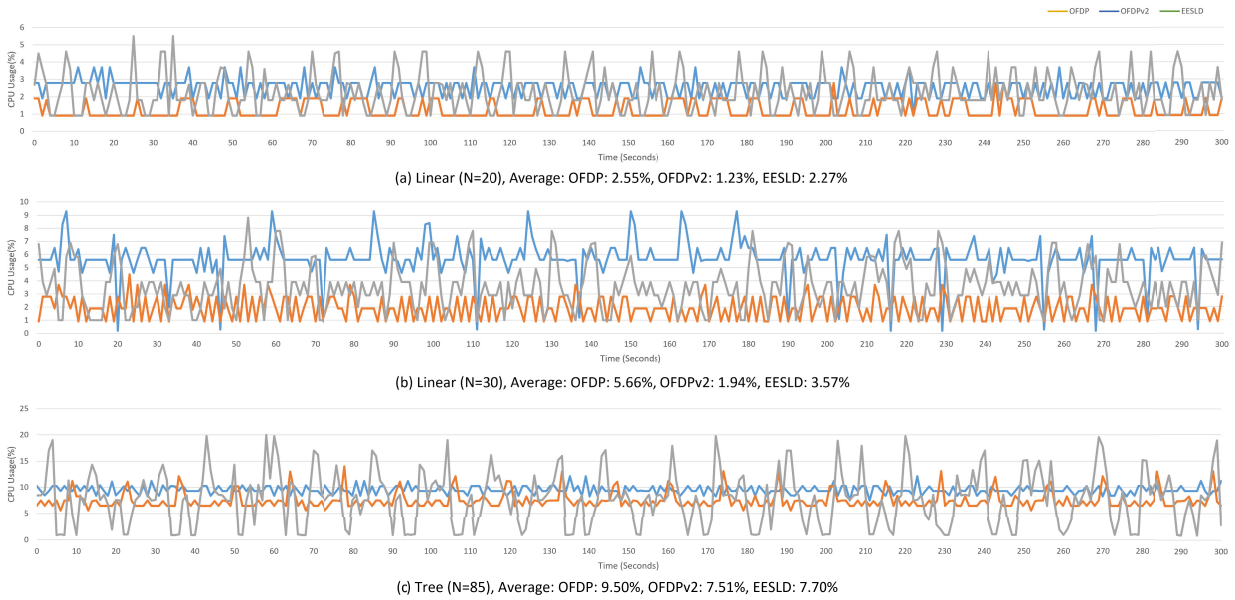


FIGURE 13. Average detection time for OSPF-based legacy link removal.

The results shown in Figure 13 indicate the average discovery and removal times for legacy links at different hop counts. The convergence time for adding a link is approximately 5.4 seconds across all the tested hop counts as the minimum link state packet is sent every 5 seconds [70]. Moreover, it takes roughly 35 seconds for the network to recover from a link failure. This timeframe is determined by the router's "dead interval" setting, which is 40 seconds, allowing for a 30 to 40 second period for routers to detect a failure [70], [71]. As the number of hops increases, the



(a) Linear (N=20), Average: OFDP: 2.55%, OFDPv2: 1.23%, EESLD: 2.27%

(b) Linear (N=30), Average: OFDP: 5.66%, OFDPv2: 1.94%, EESLD: 3.57%

(c) Tree (N=85), Average: OFDP: 9.50%, OFDPv2: 7.51%, EESLD: 7.70%

FIGURE 14. Controller CPU usage.

average time it takes to discover and remove legacy links increases slightly. Higher hop counts result in longer traversal times for control messages to reach the SDN controller, leading to increased discovery and removal times for legacy links.

The evaluation indicates that sFlow-based link discovery in the EESLD framework outperforms OSPF-based link-state advertisements for more rapid discovery and removal times for legacy links. This highlights the effectiveness and efficiency of the sFlow approach, making it a more suitable solution for discovering and monitoring legacy links in a hybrid SDN network.

C. RESOURCE CONSUMPTION

This section examines the resource consumption of the EESLD framework for SDN link discovery in hybrid SDN networks. To assess resource consumption, we considered factors such as message overhead, CPU utilisation, memory usage of the SDN controller, and CPU utilisation of switches. We employed the ''top'' command in the Linux shell to measure CPU and memory usage. The ''top'' command is a tool used to monitor processes and system status in Linux systems by providing real-time statistics on processes' performance.

1) THE MESSAGE OVERHEAD OF THE CONTROLLER

The message overhead of the controller refers to the network traffic generated and received (Packet-Out and Packet-In messages) by the controller to discover and monitor the network's SDN links. In our evaluation, we used four network topologies: Linear, Tree, Mesh, and Hybrid (star and ring), as shown in Table 4. We recorded the number of Packet-Out messages generated and Packet-In messages received by the controller over 60 seconds, starting from network initiation.

TABLE 4. Topologies and key parameters.

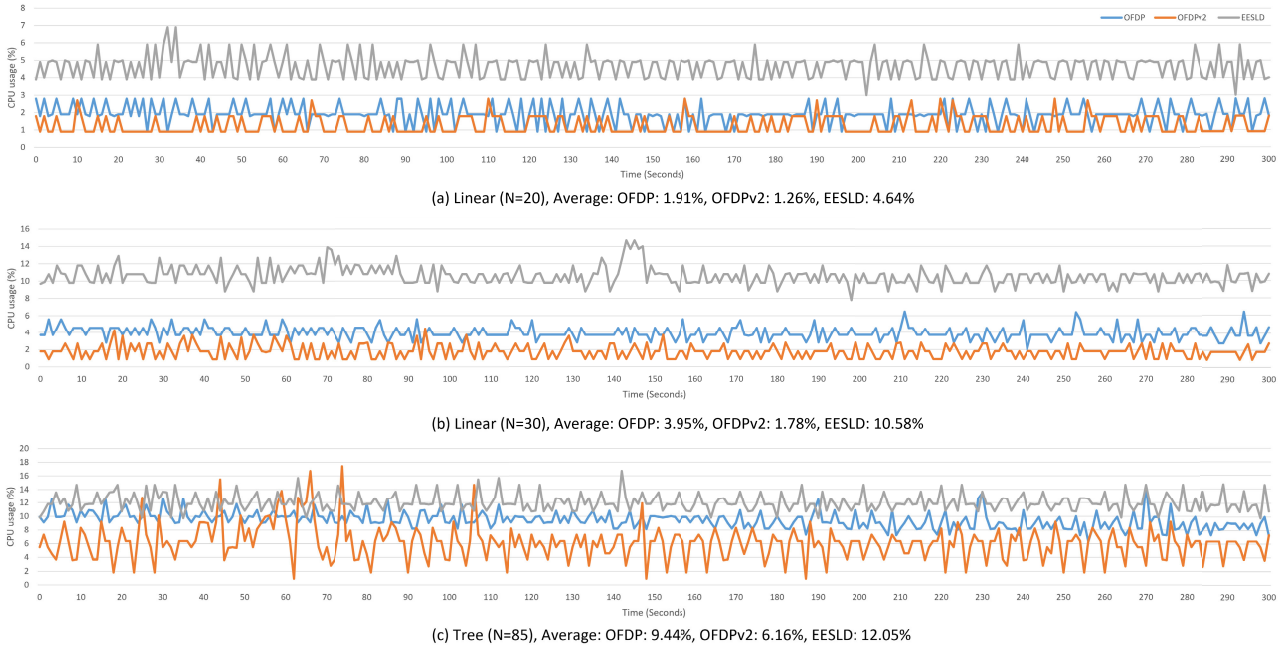
Topology	Switches	Active Ports	Links Between Switches	Hosts
Linear	20	58	19	20
Tree	15	36	14	8
Mesh	20	76	31	14
Hybrid	15	68	28	12

TABLE 5. The number of messages for discovery and maintaining topologies.

Topology	OFDP		OFDPv2		ESSLD	
	P_{IN}	P_{OUT}	P_{IN}	P_{OUT}	P_{IN}	P_{OUT}
Linear	2280	3480	2280	1200	139	120
Tree	1680	2160	1680	900	62	48
Mesh	3720	4560	3720	1200	154	114
Hybrid	3360	4080	3360	900	124	96

Table 5 compares the message overheads of the EESLD, OFDP and OFDPv2. The message overhead of the EESLD is significantly lower than that of OFDP and OFDPv2 for all network topologies.

The EESLD framework employs a single Packet-In message for each link discovery. It optimises the process by minimising the number of switches required to monitor link statuses. OFDP employs a consistent pattern of generating one Packet-Out message and receiving one Packet-In message for each switch port during every time interval. This results in a message overhead equal to twice the number of active inter-switch links in the network. An improvement in OFDPv2, reduces the number of Packet-Out messages required to maintain the network topology to equal the number of switches present in the network. However, despite



(a) Linear (N=20), Average: OFDP: 1.91%, OFDPv2: 1.26%, EESLD: 4.64%

(b) Linear (N=30), Average: OFDP: 3.95%, OFDPv2: 1.78%, EESLD: 10.58%

(c) Tree (N=85), Average: OFDP: 9.44%, OFDPv2: 6.16%, EESLD: 12.05%

FIGURE 15. Switch CPU usage.

this improvement, the number of Packet-In messages remains unchanged.

2) CONTROLLER CPU USAGE

To analyse the CPU utilisation of the SDN controller, we measured the percentage of CPU usage of the EESLD after topology construction. We used three network scales, 20, 30, and 85 switches. As shown in Figure 14, the average CPU usage of OFDP is higher for all the network scales than the OFDPv2 and EESLD because of the additional active ports. At 20 switches, OFDP exhibits an average CPU usage of 2.55%. As the network scale increases to 30 switches, the average CPU usage rises to 5.66%, and then to 9.50%, when the network scale increases to 85 switches. With OFDPv2, the average CPU usage is 1.23% for 20 switches, 1.94% for 30 switches, and 7.51% for 85 switches. By eliminating the number of Packet-Out messages, the OFDPv2 method achieves a significant reduction in CPU usage compared to OFDP. For the EESLD method, the average CPU usage is 2.27% for 20 switches, 3.57% for 30 switches, and 7.70% for 85 switches. Although the average CPU usage of EESLD is slightly higher than OFDPv2 for 85 switches, the gap reduces as the number of switches increases, indicating that EESLD is likely to outperform OFDPv2 in larger tree networks.

3) CONTROLLER MEMORY USAGE

To evaluate the memory usage of the SDN controller, we measured the amount of memory allocated to storing the network topology information after topology construction. We compared the memory usages of the EESLD, OFDPv2, and OFDP for three different network scales: 20, 30, and 85 switches.

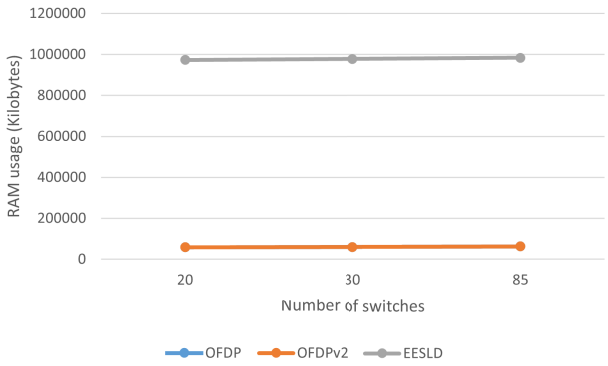


FIGURE 16. Memory usage.

As shown in Figure 16, the result reveals that the memory usages of OFDP and OFDPv2 were almost equal across all the network scales. This indicates that the implementation of OFDPv2 does not significantly reduce memory usage compared to OFDP. The memory usage of the EESLD is higher than that of OFDPv2 for all the network scales examined. Therefore, the EESLD may require more memory resources to store the network topology information. To overcome this constraint of EESLD, further investigation into optimisation techniques and memory management strategies can be considered as future work.

4) SWITCH CPU USAGE

In the final experiment, the CPU usage of switches was measured and compared for three network scales: 20 switches,

30 switches, and 85 switches. We measured the CPU usage of the Open vSwitch process to illustrate the aggregate overhead of all switches in a topology.

As shown in Figure 15, the results demonstrate that the EESLD exhibits higher CPU overhead on switches than OFDP and OFDPv2 due to the additional computational tasks involved in SDN link discovery and maintenance. For instance, in the EESLD, Open vSwitches run BFD sessions on each discovered SDN link, and some switches are required to handle port statistics requests from the SDN controller. In contrast, Open vSwitches in OFDP and OFDPv2 only need to broadcast the LLDP packet to all the ports, resulting in less strain on the switches' processing capabilities. However, the difference in switch CPU usage between EESLD and OFDP is relatively small for tree networks. This suggests that the EESLD may perform better in tree networks with a more significant number of switches.

X. LIMITATIONS AND FUTURE WORK

This section discusses EESLD framework limitations and highlights potential areas for future research.

The EESLD framework may be unable to discover indirect SDN links between OpenFlow switches separated by legacy routers lacking support for the sFlow protocol. The EESLD uses single-hop BFD, which is used primarily for directly connected routers or switches that are on the same network segment. This limitation restricts the system's capacity to fully comprehend the network topology in such scenarios. To address this limitation, future work could incorporate multi-hop BFD, which monitors links that span network segments and are not directly connected.

The EESLD framework potentially requires more memory resources for discovering and storing network topology information. This is due to the additional computational tasks involved in SDN link discovery and maintenance in the EESLD compared to state-of-the-art protocols like OFDPv2. However, with recent advancements in hardware capabilities, such as increasing memory capacity, this limitation may become less of a concern. In addition, future work could explore optimisation techniques to reduce the memory requirements of the EESLD without compromising its performance.

This study has demonstrated the significant efficacy of the EESLD framework in wired network environments. However, further investigation and improvement can be pursued by examining the potential benefits of utilising P4 programmable data planes. Integrating P4 programmable data planes into the EESLD approach would allow researchers to explore enhanced efficiency and flexibility in link discovery within SDN networks.

In addition, a further limitation of the EESLD framework is its design for wired networks, which may not directly apply to wireless networks. The link characteristics and dynamics differ significantly between wireless environments and wired networks [72]. This can result in challenges with accurately discovering and maintaining the network topology using the EESLD framework in wireless networks. As future research, it would be valuable to explore the development of link discovery mechanisms specifically designed for Software Defined Wireless Sensor Networks (SDWSN), considering the unique characteristics and challenges that SDWSN environments pose.

XI. CONCLUSION

This paper has discussed the topic of link discovery in hybrid multi-controller SDN networks. We have used the Hybrid Multi-controller SDN Network (HMSN) architecture as a case study to highlight the limitations of existing link discovery methods in dynamic and large-scale hybrid multi-controller SDN networks. Based on our analysis of existing link discovery solutions and their limitations, we have proposed an Effective, Efficient and Scalable Link Discovery (EESLD) framework. The EESLD effectively discovers direct and indirect SDN links within intra-domain and inter-domain networks. Also, EESLD selects the minimum number of switches necessary to cover all the discovered SDN links to monitor and maintain an accurate SDN link topology. EESLD classifies these selected switches based on their importance in order to determine the frequency with which these updates should be sent. EESLD uses the sFlow protocol to discover and monitor broadcast and unicast legacy links within the network. Additionally, EESLD employs the publish-subscribe messaging pattern to relay topology changes to all the relevant controllers. This approach enables controllers to promptly share their local network topology view, ensuring timely and synchronised updates and keeping all the controllers updated with the latest network topology information. The effectiveness, efficiency, and scalability of EESLD in HMSN were evaluated through simulations. The results demonstrate that EESLD effectively discovers and maintains the network topology with minimal overhead and latency. Compared to state-of-the-art protocols, EESLD exhibits superior scalability by reducing unnecessary discovery packets and minimising the load on the SDN controller and switch-controller communication channels for different topologies. In conclusion, the EESLD framework offers an effective, efficient, and scalable solution for link discovery in dynamic and large-scale hybrid multi-controller SDN networks.

REFERENCES

[1] C.-Y. Hong, S. Mandal, M. Al-Fares, M. Zhu, R. Alimi, K. N. Bollineni, C. Bhagat, S. Jain, J. Kaimal, S. Liang, K. Mendelev, S. Padgett, F. Rabe, S. Ray, M. Tewari, M. Tierney, M. Zahn, J. Zolla, J. Ong, and A. Vahdat, "B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in Google's software-defined WAN," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 74–87.

[2] A. Nehra, M. Tripathi, M. S. Gaur, R. B. Battula, and C. Lal, "TILAK: A token-based prevention approach for topology discovery threats in SDN," *Int. J. Commun. Syst.*, vol. 32, no. 17, pp. 1–26, Nov. 2019.

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

[4] L. O. Aday, C. C. Pastor, and A. F. Fernández, "Current trends of topology discovery in OpenFlow-based software defined networks," Universitat Politecnica de Catalunya (UPC), Castelldefels, Spain, Tech. Rep., 2015, pp. 1–6. [Online]. Available: http://upcommons.upc.edu/handle/2117/77672

[5] *Floodlight Controller*. Accessed: Jun. 11, 2023. [Online]. Available: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview

[6] *Open Daylight*. Accessed: Jun. 11, 2023. [Online]. Available: https://www.opendaylight.org/

[7] F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska, "Efficient topology discovery in OpenFlow-based software defined networks," *Comput. Commun.*, vol. 77, pp. 52–61, Mar. 2016.

[8] Y.-C. Chang, H.-T. Lin, H.-M. Chu, and P.-C. Wang, "Efficient topology discovery for software-defined networks," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 2, pp. 1375–1388, Jun. 2021.

[9] E. Rojas, J. Alvarez-Horcajo, I. Martinez-Yelmo, J. A. Carral, and J. M. Arco, "TEDP: An enhanced topology discovery service for software-defined networking," *IEEE Commun. Lett.*, vol. 22, no. 8, pp. 1540–1543, Aug. 2018.

[10] L. Ochoa-Aday, C. Cervelló-Pastor, and A. Fernández-Fernández, "ETDP: Enhanced topology discovery protocol for software-defined networks," *IEEE Access*, vol. 7, pp. 23471–23487, 2019.

[11] M. W. Hussain, M. S. Khan, K. H. K. Reddy, and D. S. Roy, "Extended indirect controller-legacy switch forwarding for link discovery in hybrid multi-controller SDN," *Comput. Commun.*, vol. 189, pp. 148–157, May 2022, doi: 10.1016/j.comcom.2022.03.017.

[12] D. K. Hong, Y. Ma, S. Banerjee, and Z. M. Mao, "Incremental deployment of SDN in hybrid enterprise and ISP networks," in *Proc. Symp. Softw. Defined Netw. (SDN) Res. (SOSR)*, 2016, pp. 1–7.

[13] G. Tarnaras, F. Athanasiou, and S. Denazis, "Efficient topology discovery algorithm for software-defined networks," Inst. Eng. Technol. (IET), Stevenage, U.K., Tech. Rep. 6, 2017.

[14] M. Markovitch and S. Schmid, "SHEAR: A highly available and flexible network architecture marrying distributed and logically centralized control planes," in *Proc. IEEE 23rd Int. Conf. Netw. Protocols (ICNP)*, Nov. 2015, pp. 78–89.

[15] F. Kuliesius and M. Giedraitis, "SDN/legacy hybrid network control system," in *Proc. 11th Int. Conf. Ubiquitous Future Netw.*, 2019, pp. 504–509.

[16] C. Cain. *SNMP4SDN*. Accessed: Jun. 6, 2023. [Online]. Available: https://wiki.opendaylight.org/display/ODL/SNMP4SDN

[17] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Commun. Mag.*, vol. 51, no. 7, pp. 36–43, Jul. 2013.

[18] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, 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.

[19] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 1, pp. 27–51, 1st Quart., 2015.

[20] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 623–654, 1st Quart., 2016.

[21] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab, "SDN controllers: A comparative study," in *Proc. 18th Medit. Electrotechn. Conf. (MELECON)*, Apr. 2016, pp. 1–6.

[22] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in software-defined networking (SDN)," *Comput. Netw.*, vol. 112, pp. 279–293, Jan. 2017.

[23] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Holzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, 2013.

[24] R. Amin, M. Reisslein, and N. Shah, "Hybrid SDN networks: A survey of existing approaches," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 3259–3306, 4th Quart., 2018.

[25] Y. Zhao, J. Yan, and H. Zou, "Study on network topology discovery in IP networks," in *Proc. 3rd IEEE Int. Conf. Broadband Netw. Multimedia Technol. (IC-BNMT)*, Oct. 2010, pp. 186–190.

[26] H. Zhian, M. Bayat, M. Amiri, and M. Sabaei, "Poisoning network visibility in software-defined networks: New attacks and countermeasures," in *Proc. 7th Int. Symp. Telecommun. (IST)*, Feb. 2014, pp. 635–640.

[27] J. Flathagen and O. I. Bentstuen, "Proxy-based optimization of topology discovery in software defined networks," in *Proc. Int. Conf. Mil. Commun. Inf. Syst. (ICMCIS)*, May 2019, pp. 1–5.

[28] J. Wang, Y. Tan, and J. Liu, "Topology poisoning attacks and countermeasures in SDN-enabled vehicular networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2020, pp. 1–6.

[29] O. I. Aladesote and A. Abdullah, "Efficient and secure topology discovery in SDN: Review," in *Proc. Int. Conf. Reliable Inf. Commun. Technol.*, in Lecture Notes on Data Engineering and Communications Technologies, vol. 127, 2022, pp. 397–412.

[30] A. Zacharis, S. V. Margariti, E. Stergiou, and C. Angelis, "Performance evaluation of topology discovery protocols in software defined networks," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2021, pp. 135–140.

[31] 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.

[32] H. Gebre-Amlak, G. Banala, S. Song, B.-Y. Choi, T. Choi, and H. Zhu, "TARMan: Topology-aware reliability management for softwarized network systems," in *Proc. IEEE Int. Symp. Local Metrop. Area Netw. (LANMAN)*, Jun. 2017, pp. 1–6.

[33] F. A. F. Alenezi, S. Song, and B.-Y. Choi, "CAMEL: Centrality-aware multitemporal discovery protocol for software-defined networks," in *Proc. Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2021, pp. 1–8.

[34] X. Zhao, L. Yao, and G. Wu, "ESLD: An efficient and secure link discovery scheme for software-defined networking," *Int. J. Commun. Syst.*, vol. 31, no. 10, pp. 1–18, Jul. 2018.

[35] A. Nehra, M. Tripathi, M. S. Gaur, R. B. Battula, and C. Lal, "SLDP: A secure and lightweight link discovery protocol for software defined networking," *Comput. Netw.*, vol. 150, pp. 102–116, Feb. 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1389128618307916, doi: 10.1016/j.comnet.2018.12.014.

[36] H. Tong, X. Li, Z. Shi, and Y. Tian, "A novel and efficient link discovery mechanism in SDN," in *Proc. IEEE 3rd Int. Conf. Electron. Commun. Eng. (ICECE)*, Dec. 2020, pp. 97–101.

[37] Y. Gu, D. Li, and J. Yu, "Im-OFDP: An improved OpenFlow-based topology discovery protocol for software defined network," in *Proc. IFIP Netw. Conf. (Networking)*, Jun. 2020, pp. 628–630.

[38] A. Azzouni, R. Boutaba, N. T. M. Trang, and G. Pujolle, "SOFTDP: Secure and efficient OpenFlow topology discovery protocol," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp., Cognit. Manage. Cyber World (NOMS)*, Apr. 2018, pp. 1–7.

[39] G. Tarnaras, E. Haleplidis, and S. Denazis, "SDN and ForCES based optimal network topology discovery," in *Proc. 1st IEEE Conf. Netw. Softwarization (NetSoft)*, Apr. 2015, pp. 1–6.

[40] M. W. Hussain, S. Moulik, and D. S. Roy, "A broadcast based link discovery scheme for minimizing messages in software defined networks," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2021, pp. 1–6.

[41] Y. Jia, L. Xu, Y. Yang, and X. Zhang, "Lightweight automatic discovery protocol for OpenFlow-based software defined networking," *IEEE Commun. Lett.*, vol. 24, no. 2, pp. 312–315, Feb. 2020.

[42] J. Alvarez-Horcajo, E. Rojas, I. Martinez-Yelmo, M. Savi, and D. Lopez-Pajares, "HDDP: Hybrid domain discovery protocol for heterogeneous devices in SDN," *IEEE Commun. Lett.*, vol. 24, no. 8, pp. 1655–1659, Aug. 2020.

[43] I. Martinez-Yelmo, J. Alvarez-Horcajo, J. A. Carral, and D. Lopez-Pajares, "EHDDP: Enhanced hybrid domain discovery protocol for network topologies with both wired/wireless and SDN/non-SDN devices," *Comput. Netw.*, vol. 191, May 2021, Art. no. 107983, doi: 10.1016/j.comnet.2021.107983.

[44] L. Ochoa-Aday, C. Cervelló-Pastor, and A. Fernández-Fernández, "Discovering the network topology: An efficient approach for SDN," *Adv. Distrib. Comput. Artif. Intell. J.*, vol. 5, no. 2, pp. 101–108, Nov. 2016.

[45] L. Ochoa-Aday, C. Cervelló-Pastor, and A. Fernández-Fernández, "Self-healing and SDN: Bridging the gap," *Digit. Commun. Netw.*, vol. 6, no. 3, pp. 354–368, Aug. 2020.

[46] Sandhya, Y. Sinha, and K. Haribabu, "A survey: Hybrid SDN," *J. Netw. Comput. Appl.*, vol. 100, pp. 35–55, Dec. 2017.

[47] M. Caria, T. Das, A. Jukan, and M. Hoffmann, "Divide and conquer: Partitioning OSPF networks with SDN," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, May 2015, pp. 467–474.

[48] C.-Y. Chu, K. Xi, M. Luo, and H. J. Chao, "Congestion-aware single link failure recovery in hybrid SDN networks," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 1086–1094.
[49] S. Vissicchio, O. Tilmans, L. Vanbever, and J. Rexford, "Central control over distributed routing," in *Proc. ACM Conf. Special Interest Group Data Commun.*, Aug. 2015, pp. 43–56.
[50] M. Caria and A. Jukan, "Link capacity planning for fault tolerant operation in hybrid SDN/OSPF networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–6.
[51] R. Amin, N. Shah, B. Shah, and O. Alfandi, "Auto-configuration of ACL policy in case of topology change in hybrid SDN," *IEEE Access*, vol. 4, pp. 9437–9450, 2016.
[52] T. Y. Cheng and X. Jia, "Compressive traffic monitoring in hybrid SDN," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 12, pp. 2731–2743, Dec. 2018.
[53] R. Amin, N. Shah, and W. Mehmood, "Enforcing optimal ACL policies using K-partite graph in hybrid SDN," *Electronics*, vol. 8, no. 6, pp. 1–28, May 2019.
[54] M. Ibrar, L. Wang, G.-M. Muntean, J. Chen, N. Shah, and A. Akbar, "IHSF: An intelligent solution for improved performance of reliable and time-sensitive flows in hybrid SDN-based FC IoT systems," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3130–3142, Mar. 2021.
[55] M. Ibrar, L. Wang, G.-M. Muntean, A. Akbar, N. Shah, and K. R. Malik, "PrePass-flow: A machine learning based technique to minimize ACL policy violation due to links failure in hybrid SDN," *Comput. Netw.*, vol. 184, Jan. 2021, Art. no. 107706, doi: 10.1016/j.comnet.2020.107706.
[56] A. Gämperli, V. Kotronis, and X. Dimitropoulos, "Evaluating the effect of centralization on routing convergence on a hybrid BGP-SDN emulation framework," *Comput. Commun. Rev.*, vol. 44, no. 4, pp. 369–370, 2015.
[57] M. W. Hussain, K. H. K. Reddy, J. J. P. C. Rodrigues, and D. S. Roy, "An indirect controller-legacy switch forwarding scheme for link discovery in hybrid SDN," *IEEE Syst. J.*, vol. 15, no. 2, pp. 3142–3149, Jun. 2021.
[58] J. Ali, G.-M. Lee, B.-H. Roh, D. K. Ryu, and G. Park, "Software-defined networking approaches for link failure recovery: A survey," *Sustainability*, vol. 12, no. 10, p. 4255, May 2020.
[59] D. Katz and D. Ward, "Bidirectional forwarding detection (BFD)," Internet Engineering Task Force (IETF), Fremont, CA, USA, Tech. Rep., 2010.
[60] S. Cai, J. Lin, and C. Luo, "Finding a small vertex cover in massive sparse graphs: Construct, local search, and preprocess," *J. Artif. Intell. Res.*, vol. 59, pp. 463–494, Jul. 2017.
[61] T. Opsahl, F. Agneessens, and J. Skvoretz, "Node centrality in weighted networks: Generalizing degree and shortest paths," *Social Netw.*, vol. 32, no. 3, pp. 245–251, Jul. 2010.
[62] M. Wang, B. Li, and Z. Li, "SFlow: Towards resource-efficient and agile service federation in service overlay networks," in *Proc. 24th Int. Conf. Distrib. Comput. Syst.*, 2004, pp. 628–635.
[63] InMon Corp. *sFlow-RT*. Accessed: Jun. 30, 2023. [Online]. Available: https://sflow-rt.com/
[64] *Mininet*. Accessed: Jan. 18, 2023. [Online]. Available: http://mininet.org/
[65] *RYU SDN Framework*. Accessed: Jun. 11, 2023. [Online]. Available: https://ryu-sdn.org/
[66] *Redis*. Accessed: Jun. 11, 2023. [Online]. Available: https://redis.io//
[67] *Wireshark*. Accessed: Jan. 18, 2023. [Online]. Available: https://www.wireshark.org/
[68] *Open vSwitch*. Accessed: Jan. 18, 2023. [Online]. Available: https://www.openvswitch.org/
[69] *GNS3 | The Software That Empowers Network Professionals*. Accessed: Jun. 15, 2023. [Online]. Available: https://www.gns3.com/
[70] M. Goyal, M. Soperi, E. Baccelli, G. Choudhury, A. Shaikh, H. Hosseini, and K. Trivedi, "Improving convergence speed and scalability in OSPF: A survey," *IEEE Commun. Surveys Tuts.*, vol. 14, no. 2, pp. 443–463, 2nd Quart., 2012.
[71] D. Zhao, X. Hu, and C. Wu, "A study on the impact of multiple failures on OSPF convergence," *Int. J. Hybrid Inf. Technol.*, vol. 6, no. 3, pp. 74–75, 2013.
[72] H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke, "A survey on software-defined wireless sensor networks: Challenges and design requirements," *IEEE Access*, vol. 5, pp. 1872–1899, 2017.

ISMAIL AL SALTI received the B.Sc. degree (Hons.) in computer networking from the University of Technology and Applied Sciences (UTAS), Oman, and the M.Sc. degree (Hons.) in computer networking technology from Northumbria University, U.K. He is currently pursuing the Ph.D. degree in software-defined networking (SDN) security with The University of Manchester, U.K. His research interests include software-defined networks, information security, and network security.

NING ZHANG received the B.Sc. degree (Hons.) in electronic engineering from Dalian Maritime University, China, and the Ph.D. degree in electronic engineering from the University of Kent, U.K. Since 2000, she has been with the Department of Computer Science, The University of Manchester, U.K., where she is currently a Senior Lecturer. Her research interests include security and privacy in networked and distributed systems, such as ubiquitous computing, electronic commerce, wireless sensor networks, and cloud computing, with a focus on security protocol designs, risk-based authentication and access control, and trust management.