SDN–OpenFlow Topology Discovery: An Overview of Performance Issues

Article *in* Applied Sciences · July 2021

DOI: 10.3390/app11156999

3 authors, including:

Rami Ahmad
Alpen-Adria-Universität Klagenfurt
22 PUBLICATIONS **173** CITATIONS

SEE PROFILE

applied sciences

MDPI

Review

SDN-OpenFlow Topology Discovery: An Overview of Performance Issues

Raniyah Wazirali [1], Rami Ahmad [2,*] and Suheib Alhiyari [3]

[1] College of Computing and Informatics, Saudi Electronic University, Riyadh 11673, Saudi Arabia; r.wazirali@seu.edu.sa
[2] The School of Information Technology, Sebha University, Sebha 71, Libya
[3] Department of Computer System and Technology, University of Malaya, Kuala Lumpur 50603, Malaysia; suhyeb1985@gmail.com
* Correspondence: r_a_sh2001@yahoo.com

Abstract: Software-defined networking (SDN) is an innovative architecture that separates the control plane from the data plane to simplify and speed up the management of large networks. This means the control logic has been moved from the network hardware level to the centralized control management level. Therefore, the use of the OpenFlow Discovery Protocol (OFDP) is one of the most common protocols used to discover the network topology in a data plane and then transmit it to the control plane for management. However, OFDP has various shortcomings in its performance such as exchanging too many messages between both levels (control and data), which in turn increases the load on the SDN-Controller. Additionally, since the application layer depends entirely on the network topologies plotted in the control plane, it is very important to obtain accurate network topology information from data plane. Therefore, after providing background on topology discovery protocols to the reader, we will concentrate on performance issues. The present study identifies and discuss the primary concerns involved in the complex query process, infrastructure, influencing factors, and challenges for the topology discovery process. Furthermore, this paper will present several recent studies that have overcome and enhanced these issues. In addition, open discussion and future work concerning these issues are also discussed.

Keywords: SDN; OpenFlow; OFDP; topology discovery; flow tables; control channel

Citation: Wazirali, R.; Ahmad, R.; Alhiyari, S. SDN-OpenFlow Topology Discovery: An Overview of Performance Issues. *Appl. Sci.* **2021**, *11*, 6999. https://doi.org/10.3390/app11156999

Academic Editor: Pedro Amaral

Received: 22 June 2021
Accepted: 27 July 2021
Published: 29 July 2021

1. Introduction

Software-defined networking (SDN) architecture is a promising solution to overcomes the limitations of traditional network architecture in terms of control, scalability, and management [1,2]. The basic idea of SDN is to separate the control plane from the data plane [3], where the control logic is extracted from network hardware and is centralized into an independent control plane. However, the process of separating the control plane from the data plane is not new, but it has limited to some applications such as NETCONF, SANE, and Ethane [4,5]. However, recently, this method of managing networks of various kinds has been introduced. The control plane can physically include a stand-alone SDN-Controller or multiple SDN-Controllers that collaborate among themselves and then act as a central SDN-Controller [6]. However, the standard process for separating SDN levels is illustrated in Figure 1.

Based on the open networking foundation (ONF) reference model for SDN architecture [7], the SDN infrastructure layer (data plane) includes the forwarding devices that forward packets based on a set of streaming rules configured by the control layer (SDN-Controller) [8]. The SDN-Controller also acts as a proxy between the infrastructure layer and the application layer. It translates high-level instructions at the application layer into low-level rules and then forwards them to the SDN-Switches at the infrastructure layer [9]. In the application layer, the network applications used such as adaptive routing, network

management, traffic monitoring [10,11], intrusion detection system, and intrusion prevention system [12]. Moreover, the same figure shows that these SDN layers communicate among themselves using two interfaces; southbound interface (SI) and northbound interface (NI). NI is needed to support and optimize a variety of networking applications, and it is used at the application layer to query the state of the infrastructure layer through the control layer. Examples such as the network topology, flow statistics, link bandwidth, and other information about the state of the network are queried. Furthermore, the application layer also uses it to control the infrastructure layer to implement high-level policies such as new configurations, respond to topology changes, or traffic requirements [13–18]. SDN-Controllers such as OpenDaylight [19], Floodlight [20], NOX [21], Ryu [22], Onix [23], and Beacon [24] are defined by their specific northbound application programming interfaces (APIs).

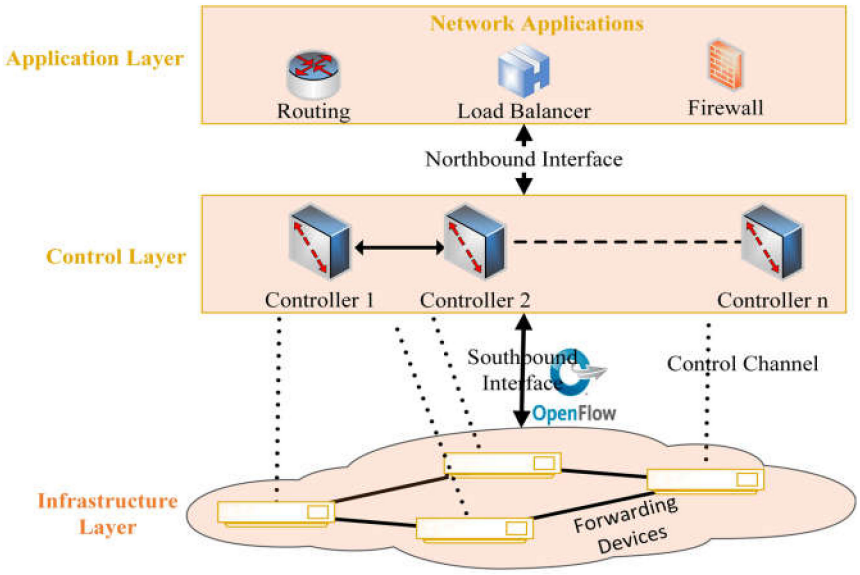

Figure 1. Standard SDN architecture [7].

On the other hand, SI acts as a link between the SDN-Controller and SDN infrastructure layer to establish settings and manage the SDN-Switches. Also, there is no standard southbound interface and there are different protocols that are used for this proposal such as ForCES [25], POF [26], OpFlex, OpenState, and OpenFlow [27]. However, OpenFlow is the most widely used protocol [28,29]. OpenFlow [28] defines a physical channel as a dedicated TCP link between SDN-Controllers (control layer) and SDN-Switches that serves as an interface for exchanging control messages using the OpenFlow protocol and also using an optional encryption mechanism called Transport Layer Security (TLS) [17,18,30,31]. Additionally, the most important role of the SDN-Controller is to maintain a consistent network topology for the infrastructure layer. The network topology maintained by the SDN-Controllers are entirely dependent on OpenFlow Discovery Protocol (OFDP) [32]. Therefore, any falsification or expiration in the data of network topology inside the SDN-Controller will directly affect the SDN-dependent services and applications [33–35]. As a result, the topology discovery service in the SDN-Controller is one of the most important.

Breitbart et al., in [36], define the network topology as a description of the physical communication relationships between network devices in networks (how SDN-Switches communicate with each other) and how the hosts communicate with them. Therefore, the topology discovery protocol consists of three main processes: SDN-Switch discovery, link discovery (between SDN-Switches), and host discovery [34]. The SDN-Controller uses link discovery to discover links between forwarding devices in the infrastructure layer. Therefore, the OFDP will be the mediator between SDN-Controller and network devices in the infrastructure layer to discover these links. OFDP uses the Link Layer

Discovery Protocol (LLDP) message format for that purpose and the SDN-Controller sends a large number of LLDP advertisements at relatively large, fixed intervals to each active SDN-Switch port in the network to discover links between SDN-Switches. The SDN-Controller uses Packet_Out OpenFlow message to send the LLDP advertisement to each active SDN-Switch port, and the SDN-Switch on the other side (data layer) will send the link information by encapsulating the LLDP packet in a Packet_In OpenFlow message. This process is cyclic every 10 s, which means that every 10 s the SDN-Controller will send several Packet_Out messages equal to the number of active SDN-Switch ports [34,35,37–39]. The results of the previous processes are a burden on the resources of the SDN-Controller and are inefficient in detecting network topology changes in a timely manner. This leads to many studies in the literature that OFDP has critical limitations in terms of performance which makes it inefficient especially for large SDN networks. Another issue related to topology discovery between SDN and non-SDN networks [40,41]. Therefore, attention to improving the performance of ODFP is an important priority that must be discussed to improve the performance of SDN technology. Different studies have been working to improve OFDP protocol in [38,42–46], but still, this domain needs more attention and research.

Moreover, in the literature, many surveys discuss SDN in general [13,47,48], and others focus on the features that have been added to SDN [49,50]. Another group discussed the adoption of SDN architecture [51–53]. Furthermore, most of the SDN OpenFlow surveys have discussed the security and scalability [3,15,29,31,51,54,55], and the authors in [56] had worked on the OpenFlow in term of performance but did not discuss the details of the topology discovery process in the SDN. Therefore, this survey focuses on the performance issues in SDN-OFDP networks. The main contributions of this paper can be summarized as follows:

- Explain in-depth how OFDP works, the factors that affect its performance, and OFDP limitations.
- A survey of the recent existing techniques found in the literature in response to enhance the performance of OFDP. The pros and cons of each technique also are highlighted.
- OFDP's open challenges and future research solutions.

The rest of the paper is organized as follows. Section 2 introduces a brief background about SDN architecture and OpenFlow southbound interface. Section 3 presents the topology discovery in SDN networks and its de facto OFDP, performance issues in OFDP, and the current proposals to enhance the performance of OFDP. Section 4 discusses the pros and cons of the proposals for OFDP and Section 5 concludes the paper.

2. Topology Discovery in SDN Networks

As we discussed in the introduction, the SDN technology is based on separating the control layer from the data layer (forwarding devices) to facilitate network scalability and management operations. However, the process of discovering the network topology relies on cooperation between the SDN-Controller and forwarding devices layers, and the responsibility lies primarily with the control layer. The control layer topology discovery task mainly consists of three main operations: SDN-Switch discovery, link discovery (i.e., links between SDN-Switches), and host discovery [33,34] as illustrated in Figure 2.

Since we need OpenFlow to act as a data pipeline between the control layer and the data layer, the loading of the topology discovery process is entirely on the controller layer. Therefore, the OpenState protocol has been proposed as a modification of OpenFlow that attempts to divide the load of the topology discovery process between SDN-Controller and SDN-Switches. As a result, OpenFlow SDN-Switches can be directly programmed, allowing them to implement redirection rules without relying solely on the remote controller. As a future improved release of OpenFlow, OpenState has not yet been deployed [57].

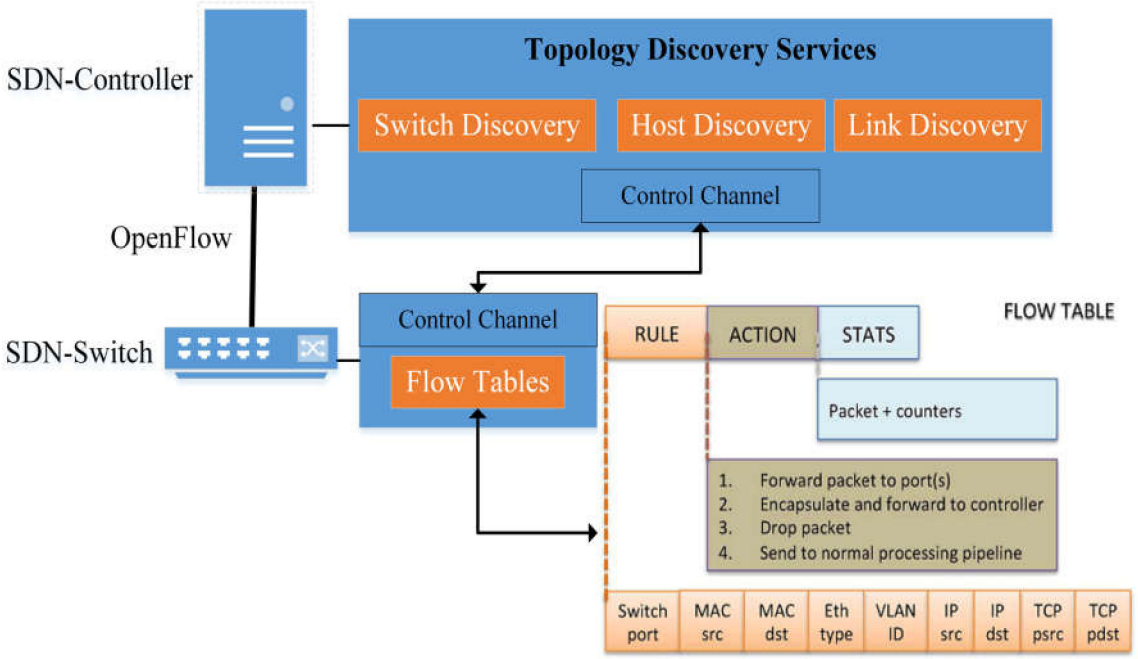

Figure 2. Topology discovery processes [29].

The OpenFlow architecture combines two layers through the use of flow tables in the forwarding devices layer. Moreover, each flow table entry contains three features (rule, action, and statistics) [15] and each flow table contains action fields that are linked with each flow entry. These flow tables data and commands are transmitted between two layers by using a control channel. In detail, the use of flow table and control channel techniques will be discussed in Sections 2.3 and 2.4.

However, the topology discovery services are discussed in detail as the follows.

2.1. SDN-Switch Discovery

According to [58] version 1.5.1, each SDN-Switch must consist of a set of tables, namely, a flow table, a matching table, and a missing table, as well as a control channel to monitor the changing flow in the different SDN-Switches via SDN-Controller. Therefore, in the SDN-OpenFlow, routing processes are based on standard flow tables rather than destination (i.e., IPs or MAC) as in the traditional networks. The flow table is used in conjunction with the logical data structure where packets are processed based on the list of priority entries in these flow tables. In each flow entries for up to 15 fields can be stored in OpenFlow version 1.10 [57], 5 of which are mandatory and the rest are optional. The matching, action, priority, timeout, and counter are the most common fields. Moreover, the SDN-Controller pins these flow entries to the flow table in two approaches: reactive and proactive, and the SDN-Controller determines which one is based on the occurrence of some events. When network activity starts, the SDN-Controller in a reactive approach does not initialize the flow table with any rules. The SDN-Controller will insert rules into the flow table whenever data arrives at switches while the network is running. For the proactive approach, the SDN-Controller will pin the flow entries into the flow table beforehand, when the network is started. The choice of rules is essential in optimizing network performance, particularly in large-scale networks. When packets reach at a switch while the network is running, the incoming packet flow matches the flow entries in the flow table. If no match is found, the SDN-Switch will call the SDN-Controller to request that entries to permit the packet to reach its destination. This includes frequent connections between the SDN-Controller and the SDN-Switch as well as delays before the packet can be transmitted to the next hop. The proactive technique was implemented to reduce the amount of time SDN-Switches and SDN-Controllers had to communicate.

In addition, each OpenFlow SDN-Switch is configured with the IP address and the TCP port number of the SDN-Controller. Thus, to join the network, the OpenFlow SDN-Switch creates a TCP session using a three-way handshake (SYN, SYN/ACK, ACK) to initiate the communication with the SDN-Controller. Next, the SDN-Controller sends an OFPT_Features_Request message to the SDN-Switch requesting its current configuration as Media Access Control address (MAC address) and network interfaces. Then, the SDN-Switch will reply with an OFPT_Features_Reply message which contains the requested information. The SDN-Controller stores and uses such information for future network management tasks including re-processing of topology discovery [39]. Figure 3 presents the process of establishing SDN-Switch discovery in SDN networks.

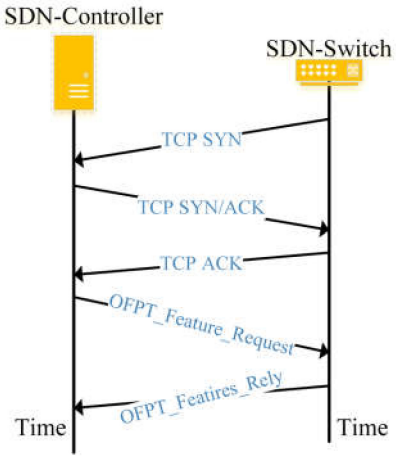

Figure 3. Creating the SDN-Switch discovery process.

After the process of establishing OpenFlow SDN-Switch connection, flow tables will contain the SDN-Switch header information and actions (responsible for giving commands). As illustrated in Figure 2, when the packets are streamed from the data plane to the SDN-Switch, each incoming packet will be checked against the flow tables, when the packet's matching header meets one in the pipeline flow tables, the rules related with the flow entry will be triggered. For each successful match between incoming packet and flow table entries, the counter field will be increased. When the packet flow reaches the input flow table (ingress) and a match is made against flow entries, if no match occurs, it moves to the next flow table using GoTo-Table instruction and then again performs a match with flow entries. This process continues seamlessly until all flow tables are finished, and therefore, the packet will be treated as miss_flow if it does not get a match in one of the flow tables. According to the instructions in the miss_flow entry, it either drops the packet or resends it to the other flow table [56]. The packet flows via an OpenFlow SDN-Switch in both directions (ingress and egress) are illustrated in Figure 4.

In the case of the network traffic being huge and complex, many unknown stream packets will arrive at the forwarding node, thus, it will produce a large number of Packet-In messages. On the other hand, sending a flow request (Packet-In message) to the SDN-Controller for each unknown packet will confuse the SDN-Controller because the SDN-Controller has to compute the forwarding rules for each new packet and then install it to the flow tables in all the data forwarding nodes (SDN-Switches). This high volume of traffic and computational overhead will cause SDN-Controller overhead and increase the time it takes for flow rules to be placed, affecting network efficiency and scalability [1,2].

Moreover, [59] claimed that the data center with a 4 K server can handle up to 200,000 flows per second. Another study found that the average flow width was roughly 20 packets per flow, with latency between flows less than 30 milliseconds [60]. These costs are very high, but the memory available to hold sending entries is limited. However, the entries for typical OpenFlow SDN-Switch flow tables were stored in Ternary Content Addressable Memory (TCAM), a type of high-speed memory that allows looking up a

continuous flow entry in a clock cycle (1). Although the TCAM search is fast, its capacity is limited to a few thousands of entries [61]. On another hand, Increasing the TCAM size raises additional issues such as cost, and will require high power consumption. Therefore, the researchers tried to optimize the flow schedule to take full advantage of it.

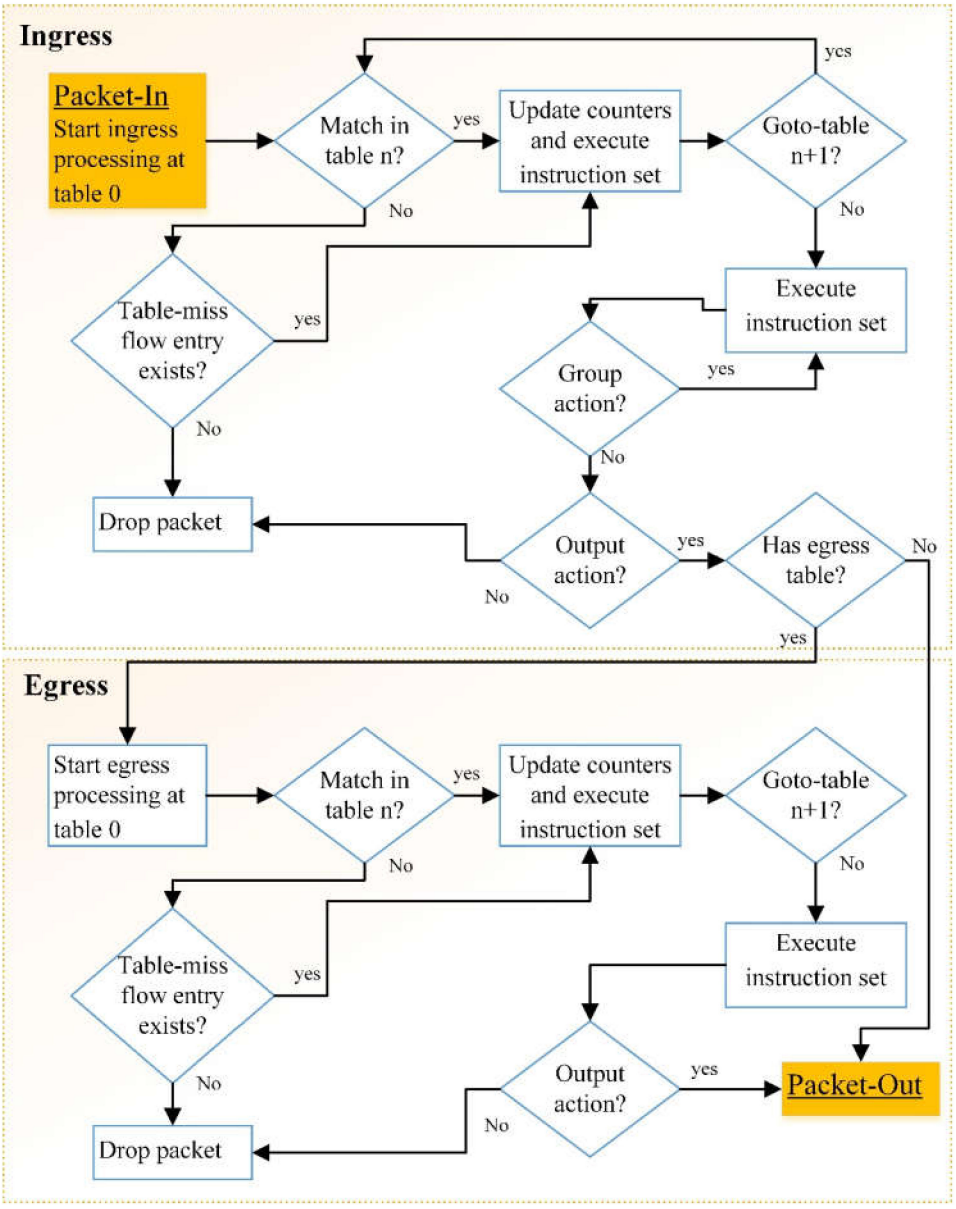

Figure 4. Flowchart showing details of packet flow via OpenFlow SDN-Switch [58].

2.2. Host Discovery

In the process of communicating and exchanging packets (in and out) in OpenFlow protocol, and when an OpenFlow SDN-Switch receives a packet that does not match any flow rule in its flow table. In this case, one possibility is that a new host is connecting to the network. The new host starts sending the packets to SDN-Switch, then the OpenFlow SDN-Switch will encapsulate that package with a Packet_In message and send it to the SDN-Controller. The SDN-Controller will in turn use this Packet-In message to discover the hosts on the network, then the SDN-Controller will extract the host's location (i.e., to which SDN-Switch port it is connected to), host's IP address, and its MAC address from Packet_In messages [32].

2.3. Link Discovery

The goal of the link discovery process is to discover the existing links between connected OpenFlow SDN-Switches and also to efficiently detect changes to the network topology. Additions and deletions are among the most common examples of network topology changes. Link deletion occurs when an existing link is removed (physically) or access fails due to other reasons. The link deletion also occurs when an existing SDN-Switch is removed or failed to access for various other reasons as well. Moreover, the processes of link additions will be similar to the processes of deletions [38]. In all of these processes, the occurrence of link changes is directly related to the SDN-Controller. Therefore, given the importance of this part in our topic, further discussion will be given in Section 2.2.

2.4. Link Discovery Protocol

As we noted in the introduction that there is no standard protocol in SDN networks to discover the links between SDN-Switches, and most of the existing SDN-Controllers use Link Layer Discovery Protocol (LLDP) [33,38,39] such as OpenDaylight [19], Floodlight [20], POX [62], Ryu [22], Beacon [24], Cisco Open SDN-Controller [63], and Open Network Operating System (ONOS) [34,38,43] to discover these links. LLDP is considered a layer-2 (Data link) protocol that is used by network devices to their identity, capacities, and neighbors' devices on a Local Area Network (LAN) based on IEEE-802. Moreover, each LLDP discovery message is embedded in a layer-2 frame, which is a type of Ethernet and the Data Unit called (LLDPUD) [56]. The data obtained by LLDP is placed into the management information database of the SDN-Switches, which can then be queried while crawling the network's nodes to retrieve the network topology using a network management protocol.

At a later stage, the NOX SDN-Controller [21] implemented the LLDP development process to improve the discovery of the link between SDN-Switches and created the first version of the OFPD protocol [64]. OFDP does not depend on a centralized control such as LLDP (i.e., switches send and receive LLDP advertisements autonomously). It is a request-response discovery protocol that can send a Packet-In message to the SDN-Controller to receive the discovery information collected. Nevertheless, OFDP uses the LLDP packet format with few modifications and operates in a slightly different way than LLDP protocol for compatibility with the SDN architecture, where the control logic is central to the SDN-Controller. Therefore, OFDP SDN-Switches do not initiate LLDP advertisements but the SDN-Controller has full control over the link discovery process. Table 1 presents the main differences between LLDP and OFDP protocols.

Table 1. Similarities and differences between LLDP and OFDP protocols [64].

Features	LLDP	OFDP
Type of Ethernet frame	LLDP's EtherType = 0 × 88cc	OFDP's EtherType = 0 × 88cc
Destination address of the frame	bridge-filtered multicast MAC (01:80:C2:00:00:0E)	normal multicast MAC (01:23:00: 00:00:01)
Mode of operation	Advertisement only	Advertisement only
What will the SDN-Switches do with advertisements?	SDN-Switches will not forward LLDP advertisements	SDN-Switches will forward OFDP advertisements
SDN-Switches' neighbor table	SDN-Switches that support LLDP build a table for directly connected neighbors	OpenFlow SDN-Switch does not keep any information about its directly connected neighbors
How is the topology obtained?	By crawling the neighborhood tables of SDN-Switches	By inferring the information from LLDP Packet-In messages

The SDN-Controller initiates the discovery process in OFDP by sending an LLDP discovery announcement encapsulated in a Packet_Out message to the forwarders (parent SDN-Switches) that are directly associated with OpenFlow using a multicast address. When the forwarder device receives the announcement message, it floods all of its ports with

an LLDP discovery announcement, and the only SDN-Switch that supports OpenFlow updates its OFDP table.

To illustrate how OFDP uses LLDP advertisement messages, the LLDP considers a layer 2 frame that consists of a header and a payload [39] as shown in Figure 5. In the header portion of the frame, the Ethertype field is set to 0x88cc, and the destination MAC address field is set to a multicast address as we discussed in Table 1. The Ethertype field is used by OpenFlow SDN-Switches to distinguish LLDP frames from others. The payload portion is called LLDPDU which is shaded grey. The payload consists of a number of fields with a Type-Length-Value (TLV) structure and ends with the "End of LLDPDU TLV" field. Some TLV fields in LLDPDU are mandatory while the others are optional. The mandatory fields contain the information that the SDN-Switch wants to advertise to its neighbor which is: Chassis-ID (which is a unique switch identifier), Port ID (which is its egress port), and time to live.

Preamble	Destination MAC	Source MAC	Ethertype	Chassis ID TLV (mandatory)	Port ID TLV (mandatory)	Time to live TLV (mandatory)	Optional TLVs	End of LLDPDU TLV	Frame check seq.
	Multicast address	MAC address of the sending port	0x88cc	Local switch ID	Sending Port ID				

Figure 5. LLDP frame format used in OFDP [39].

Therefore, to illustrate the OFDP function, we explain the cooperation between the ODFP sections based on each other, as illustrated in Figure 6. The SDN-Controller sends every specified period (i.e., 10 s) an LLDP packet encapsulated with a Packet-Out message to each active port in each SDN-Switch [38]. From Figure 6, the SDN-Switch (s1) has three active ports, which means three LLDP packets and each of these packets has Port-ID TLVs and a Chassis-ID configured accordingly. In OpenFlow protocol, if the SDN-Controller wants to send a packet to the OpenFlow SDN-Switch it will encapsulate it in a Packet-Out message. The Packet_Out message structure contains a field called the instruction field. This field is responsible for deciding what the SDN-Switch should do in that packet. Therefore, the LLDP packet is obtained from all ports in SDN-Switch except for the one connected to the SDN-Controller. This port communicates with SDN-Controller via an OpenFlow Packet-In message. Moreover, it is used to collect and aggregate all the port information in each SDN-Switch into a single "Packet-In" message and sends it to the SDN-Controller according to a proactive rule installed in the flow tables of all SDN-Switches. In the Figure 6 scenario, the LLDP packet on port-ID 1 is sent from SDN-Switch (s1) and received by SDN-Switch (s2) through port 1 as well.

Moreover, the Packet_Out message has a field called instruction which is configured to forward the encapsulated LLDP out of the corresponding port on the SDN-Switch [39,42]. Next, the SDN-Switch (s1) will, in turn, receive Packet-Out messages, de-decompile the LLDP packet from the Packet-Out message, and forward only the LLDP packet that has exited from each matching port (based on the port's MAC address in LLDPDU). When SDN-Switch (s2) receives an LLDP packet, it will parse the LLDP packet, write its SDN-Switch Chassis-ID, and add Port-ID (i.e., the port through which the SDN-Switch received the LLDP packet). Then, SDN-Switch (s2) will encapsulate the LLDP packet in a Packet_In message and send it to the SDN-Controller. The SDN-Controller will in turn parse the Packet_In message and discover the new links represented by the mapping between s1 and s2 [65]. However, the same method is repeated to discover the remaining links in the network [39].

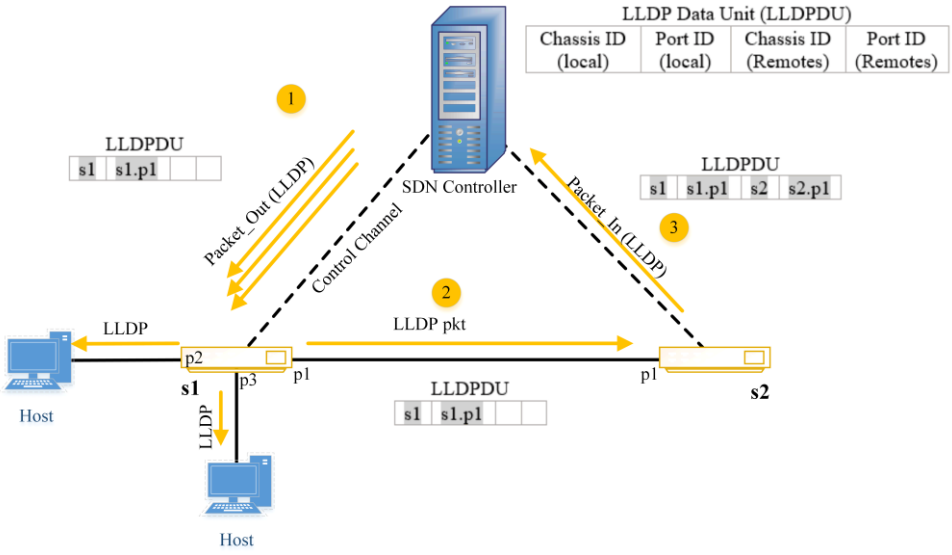

Figure 6. OFDP protocol methodology [39].

2.5. Control Channel

Each OpenFlow Logical SDN-Switch is connected to an OpenFlow SDN-Controller through the OpenFlow channel. The SDN-Controller installs and maintains the SDN-Switch using this link, collects events from the SDN-Switch, and transmits data from the SDN-Switch to the SDN-Controller. The SDN-Switch's control channel can handle a single OpenFlow channel with a single SDN-Controller or several OpenFlow channels with multiple SDN-Controllers that share SDN-Switch management. In addition, the communication connection between both the data stream and the OpenFlow channel is managed independently, but it must be liable to the OpenFlow SDN-Switching protocol rule. Moreover, the OpenFlow channel is normally performing over TCP and is secured using TLS, [58].

Furthermore, if an SDN-Switch becomes connected to a group of SDN-Controllers, the SDN-Controllers status update should be sent to the SDN-Switch with only one SDN-Controller and the other SDN-Controllers in 'standby' mode if the first controller stops. Additionally, the SDN-Switch should provide an improved Controller-Status indication for all SDN-Controllers when the OpenFlow channel is reconnected. Moreover, If the SDN-Switch loses connectivity to all SDN-Controllers for various reasons, including echo request timeouts, TLS session timeouts, or other disconnects, it should switch to "Fail-Safe Mode" or "Fail-Standalone Mode", depending on the SDN-Switch design and configuration. The only difference in the SDN-Switch's behavior in Fail-Safe Mode is that packets and messages addressed to the SDN-Controllers are dropped. In Fail-Safe Mode, flow entries should terminate according to their timeouts, while in "Failure-Standalone", the SDN-Switch uses the OFPP_NORMAL reserved port to process all packets. In other words, the SDN-Switch acts like an old Ethernet SDN-Switch or router. Additionally, the SDN-Switch can use flow tables in any way it wants when in "Fail-Standalone" and can delete, add, or edit any flow entry. By the way, only Hybrid SDN-Switches usually have a Fail-Standalone mode [58].

Moreover, according to [66,67], a slow control channel significantly reduces data layer throughput and response time in addition that it will threaten network availability. Therefore, several recent works have been proposed to help maintain OpenFlow availability.

2.6. Performance of the Link Discovery (OFDP)

One of the most important services of the SDN-Controller is to provide an updated and comprehensive network topology under its control. All network applications depend entirely on the network topology received from the SDN-Controller. Thus, any issues with

the performance of this service in the SDN-Controller will negatively affect the performance of the entire SDN [6,46,68,69]. These performance issues are more noticeable in the case of dynamic and large networks.

Regarding the topology discovery service, several experiments were performed on different SDN-Controllers with only one discovery unit running. The results showed that when the number of SDN-Switches (i.e., network size) reaches a certain limit, there is a significant increase in the CPU usage of the SDN-Controller and a significant decrease in the network performance [39,69,70]. In [71], the authors empirically evaluated the performance of the OpenDaylight and the ONOS SDN-Controllers in terms of updating the topology discovery process. The authors used the topology discovery time and throughput as performance metrics, and their results showed that the ONOS performed better in terms of network throughput in the event of topology changes while the OpenDaylight outperformed ONOS in topology discovery time. Therefore, in this subsection, we will analyze how the network size affects the SDN-Controller performance during the network topology discovery process.

2.6.1. OFDP in Huge and Dynamic Environments

However, there is not much research analyzing the OFDP performance on the SDN networks and most of the researchers in OpenFlow performance analysis have concentrated on SDN-Controller types such in [55,72–74]. Therefore, ODFP performance analysis would be appropriate to open up this issue to the researchers to search in these domains.

In this work, we will discuss the performance issues of topology discovery in the OpenFlow protocol and its impact on the SDN-Controller, where OFDP will have a major role.

In [69], the authors investigated the OFDP performance for transport networks (i.e., backbone service provider networks) and found that when carrier rank requirements are met, the transport networks should recover from link failure within a maximum of 50 ms. Thus, to achieve a 50 ms recovery time for OpenFlow-based carriers, the topology discovery service at the SDN-Controller should run approximately every 10 ms. Therefore, the SDN-Controller must check hundreds of LLDP Packet_Out messages per active SDN-Switch port every second to detect one direction per link. Moreover, it also must receive and process two hundred LLDP Packet_In messages every second for each link and end-to-end tunnel. In addition, transport networks have hundreds of links and thousands of tunnels. This means that the SDN-Controller has to handle millions of messages per second just to monitor the health of the network. This undoubtedly imposes a large load on the SDN-Controller and also a large overhead on the control network, especially for in-band control channels. In addition, another type of network environment where OFDP shows performance issues is multi-tenant cloud data centers. In such environments, the network topology is dynamic, because the tenant can build and modify their network; They can add and remove SDN-Switches or links at any time. This means that the topology can dynamically and continuously change [38]. Hence, the SDN-Controller must be efficient enough to maintain an updated network topology. However, for OpenFlow networks where OFDP is the link discovery protocol, the SDN-Controller only discovers the topology at periodic, constant, and relatively long intervals (floodlight controller for example every 15 s). Therefore, the SDN-Controller only realizes the new topology changes in each discovery round, and if an error occurs, the error correction needs to wait for the next topology discovery round which is too long [75]. As a result, application-level network applications such as routing will use the wrong network configuration until at least the next topology round.

2.6.2. OFDP Performance Metrics

As we discussed earlier, three main entities participate with each other in the OFPD protocol to get work done: the SDN-Controller, SDN-Switches, and control channels between the SDN-Controller and SDN-Switches. The SDN-Controller will send, receive,

and process messages related to the discovery process. Likewise, SDN-Switches will receive, send, process messages, and all of these messages will use the control channel as arguments. Ultimately, these processes are an overhead on all participating entities. The overhead incurred by OFDP consists of the OFDP-connection to the connector (control channel) from one side and processing overheads on the SDN-Controller and SDN-Switches on the other side. Until this moment, there are no performance metrics that are adopted by any of the standard organizations to measure the performance of the topology discovery service in SDN, but there is some research suggesting and using some of the commonly recognized performance metrics. Therefore, we will discuss the performance metrics used in these studies and suggest some others for their importance as follows:

1. The number of packets sent and received by the SDN-Controller

The authors in [39,42,45,76] used the number of packets sent and received by the SDN-Controller as a performance metric. To discover the links between SDN-Switches, in each discovery round the SDN-Controller sends several LLDP Packet_Out (P_{OUT}) messages equals to the number of active SDN-Switch ports in the network and will receive a number of Packet_In (P_{IN}) messages equals to twice of the number of links as shown in Equations (1) and (2).

$$P_{OUT} = \sum_{i=1}^{N} P_i \quad (1)$$

$$P_{IN} = 2L \quad (2)$$

where N is the number of SDN-Switches and P_i is the number of ports of DS-Switch i, L is the number of links between the SDN-Switches in the network.

As illustrated from Figure 6, the SDN-Controller will send three LLDP Packet_Out messages equal to the number of active ports on SDN-Switch (s1) and it will receive one LLDP Packet_In message to discover the one-way link from s1 and s2. The SDN-Controller also needs to send another three LLDP Packet_Out messages to s2 to discover the one-way link from s2 to s1 and it also needs to receive an LLDP Packet_In message. In total, the SDN-Controller will send six LLDP Packet_Out messages (i.e., number of active ports on s1 and s2) and two LLDP Packet_In (i.e., double of the number of links) messages to discover the link between s1 and s2.

Moreover, the authors in [39] performed three experiments with different topologies as shown in Table 2. The letters 'd' and 'f' donate the depth and fan-out parameters in tree topologies as well as the letter 'm' refers to the number of SDN-Switches in linear topologies. The experimental results reveal that the number of LLDP Packet_Out messages is equal to the number of active ports regardless of the topology type as previously mentioned in Equation (1).

Table 2. Number of Packet_Out messages in different topologies [39].

Topology Number	Topology Type	Topology Parameters	Number of SDN-Switches	Number of SDN-Switches	Number of Packet_Out
Topology 1	Tree	d = 4, f = 4	85	424	424
Topology 2	Tree	d = 7, f = 2	127	380	380
Topology 3	Linear	m = 100	100	298	298

There are two types of SDN network topologies; Linear and tree [77]. In a linear structure, each SDN-Switch is associated with a single host, and in a tree, the SDN-Switches are arranged like tree branches as well as terminal branches associated with hosts. It is the only structure without loops, and the tree topology has the highest throughput with OpenDaylight, which is approximately 13 gigabits per second (Gbps) [77]. However, the drawbacks of the linear structure are the occurrence of disturbances in the network if the network element fails or smashes, and the troubleshooting difficulty is high and time-consuming.

2. Average CPU Utilization of SDN-Controller

In [38,39,75], the authors use this metric to measure the extent to which OFDP is using the SDN-Controller to obtain the topology. The SDN-Controller uses its CPU to create, send LLDP Packet_Out messages, and process LLDP Packet_In messages. The average CPU utilization increases when the number of packets sent and received by the SDN-Controller increases.

3. Accumulative CPU Utilization of SDN-Switches

The authors in [42] used this metric to measure the extent to which OFDP is using the CPU for SDN-Switches. SDN-Switch is an essential part of discovering the topology. The SDN-Switch receives LLDP Packet_Out messages from the SDN-Controller and sends them to its active ports. As a result, the number of packets sent or received by the SDN-Switches will also increase the CPU utilization ratio.

4. Bandwidth Consumed by OFDP

As described in the OFDP methodology, there are two types of connections: between the SDN-Switches themselves and between SDN-Switches and the SDN-Controller. This metric is determined by the size of the exchanged OFDP packet to maintain the topology. In [39,42], the authors used this metric to evaluate OFDP. Thus, this bandwidth can be especially important for measuring the performance of large networks and in-band control channels.

5. Learning Time

Some researchers [38,42,76] have used this metric to evaluate OFDP and the performance of their topology discovery. Learning time is the time the SDN-Controller needs to learn about topology changes. The discovery process will be repeated every discovery interval. The discovery interval is the time interval between two discovery rounds. The problem is that when a topology change occurs, the SDN-Controller will wait for the next discovery round to learn about new topology changes. This means that learning time is at least equal to the discovery interval.

2.7. Challenges of the Link Discovery

As we discussed in Section 2.4, on each fixed time interval (10 s) the SDN-Controller sends an LLDP packet encapsulated with a Packet-Out message to each active SDN-Switch port in the network. This discovery mechanism could present serious performance issues to SDN networks, especially of large networks. Therefore, we will summarize the OFDP link discovery challenges as follows:

1. Overhead to SDN-Controller and Control Channel

This point of the challenge has been clarified in details in the OFDP performance subsection, and therefore we will avoid re-explaining it

2. Inefficient Link Failure Detection

In most SDN-Controllers, each round of detections is performed approximately every 10 s and the SDN-Controller will become aware of new topology changes [44]. This is a very long process for dynamic network environments where the changes to the topology occur frequently over a short period [38]. This greatly affects the operation of network applications that depend entirely on the SDN-Controller's network topology [78]. For example, if the learning time is long and there is a link failure on router links, the routing application will still redirect traffic on that path based on a legacy network topology, which means this will cause many packets to be dropped. Figure 7 shows the relationship between learning time and topology changes. When a topology change occurs after the first discovery round, the SDN-Controller will wait for the next round to re-detect changes to the topological structure, which is too long. On the other hand, there is a suggestion to reduce the interval of discovery changes, but this will cause the number of the Packet_Out messages to increase significantly, which in turn will increase the load on

the SDN-Controller and uses more bandwidth in the process of discovering the topology. In addition, an increase in the discovery interval means fewer LLDP messages and less overhead, but also means more time required to learn about new changes.

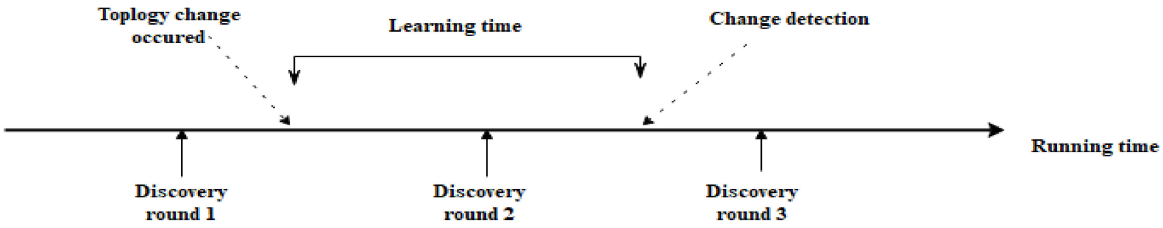

Figure 7. Learning time between rounds of topology discovery.

Regarding large and dynamic network environments as we discussed in Section 2.6.1 these challenges appear clearly, and based on performance metrics, we expect that OFDP performance will perform as shown in Table 3. The table shows that the value of some performance metrics is considerable and cannot be ignored [38,69].

Table 3. OFDP performance in large and dynamic networks.

Performance Metrics	Description	Outcome
Number of packets sent and received by the SDN-Controller	• The number of packets sent equals the number of active SDN-Switch ports • The number of packets received equals twice the number of links	Considerable Reasonable
Number of packets sent and received by each SDN-Switch	• The number of packets sent equals the number of its active ports that connected to SDN-Switches • The number of packets received equals the number of its active ports	Reasonable
Average CPU utilization of SDN-Controller	• CPU overhead added by OFDP to SDN-Controller	Considerable
Accumulative CPU utilization of SDN-Switches	• CPU overhead added by OFDP to SDN-Switches	Reasonable
Bandwidth consumed by OFDP for in-band control channels	• How much does OFDP use from control channel bandwidth?	Considerable
Learning time	• How long does it take for OFDP to learn about topology change?	Considerable

3. Security Issues

Adding security to the topology discovery processes in OFDP also poses a new challenge in terms of Quality of Services (QoS). Secure OFDP is a great idea to protect the real-time topology from attacks such as inserting malicious rules at SDN-Switch, denial of services at SDN-Controller, and man-in-the-middle in control channel [52], but it causes other issues related to the performance of the device and time-consuming. Therefore, given the contrast of the two trends between network performance and security in sensitive issues (network topology discovery), the issue of balancing between them is also important. Therefore, this makes the door of research open to researchers to balance security and performance according to the needs of the network.

However, with challenges such as these, the SDN discovery topology protocol needs to provide the SDN-Controller with a real-time view of the network topology to meet the application and dynamic routing Quality of Service (QoS) demands.

3. Recent SDN Topology Discovery Performance Studies

In the literature, the studies that discuss the issue of performance in OFDP or OpenFlow are considered rather scarce, and most of the studies dealing with and maintaining OpenFlow security [33–35,79–81]. Therefore, in this paper, we will focus on OpenFlow, and in particular on OFDP protocol, along with the factors that have a direct impact on its performance. This section provides summaries of this research in detail as follows:

3.1. Link Discovery Improvement Algorithms

In this subsection, we will discuss the proposed solutions that are related to OFPD link discovery. These proposals can be categorized into two domains based on their procedural release (event or periodic). Moreover, a comparison of these proposals is provided in Table 4 as well.

3.1.1. Periodic

This domain is named periodic because the discovery process is carried out periodically for every period. The authors in [39,42,43,45] used this type as we will discuss.

In [39], the authors proposed a new approach called OpenFlow Discovery Protocol version 2 (OFDPv2) to enhance the OFDP performance. OFDPv2 reduces the number of LLDP Packet_Out messages to only one LLDP Packet_Out message per SDN-Switch instead of per active port. They proposed two versions of OFDPv2. OFDPv2-A for SDN-Controller and OFDPv2-B for SDN-Switch. In OFDPv2-A, the SDN-Controller will install an additional set of flow rules in each SDN-Switch using the OFPT_FLOW_MOD message to forward the LLDP packets from each port on the SDN-Switch. The OFPT_FLOW_MOD message is used by the SDN-Controller to process the flow tables of the OpenFlow SDN-Switches. It can add, update or delete flow entries from the flow tables of OpenFlow SDN-Switches. However, added rules consume a lot of Ternary Content-Addressable Memory (TCAM) which is already a scarce resource in SDN-Switches. In OFDPv2 B, they did not install flow rules on SDN-Switches but they send an action list (i.e., a set of instructions to forward LLDP packet for each port) with the LLDP Packet_Out message. However, this makes OFDPv2-B withstand a large bandwidth, especially for in-band control channels. These processes are illustrated in Figure 8.

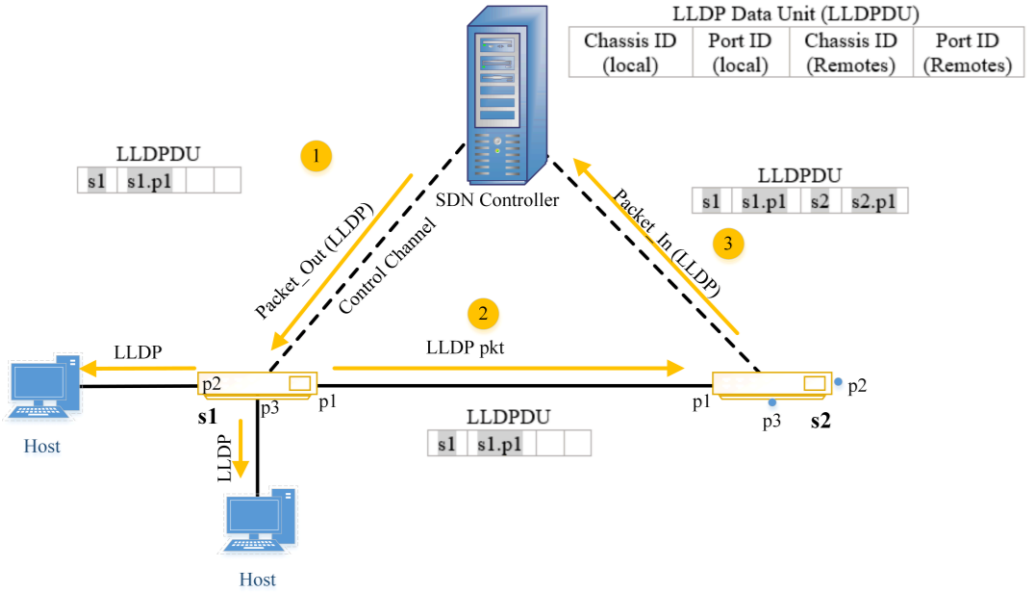

Figure 8. Methodology of OFDPv2.

Table 4. Comparison between different proposed approaches to enhance OFDP.

Proposal	Methodology	Location of Logic	Operation Mode	Advantage	Disadvantage
OFDPv2	Merge Packet_In messages from each port to each SDN-Switch	Controller	Periodic	(1) Reducing CPU overhead. (2) Reducing the bandwidth overhead on the control channel.	It works periodically, and this may introduce unnecessary discovery traffic
sOFTDP	Triggering topology discovery service by link failure events monitored by Bidirectional Forwarding Detection.	Switches	Event	(1) Reduce learning time. (2) Reduce adaptation time. (3) Reduce CPU overhead.	It restricts the controller's ability to collect statistical data on discovery traffic
ForCES	Delegating the logic of topology discovery to the SDN-Switches.	Switches	Event	(1) Reduce the learning time of link changes.	It is only applicable for ForCES as a southbound API.
ESLD	Reducing messages between the SDN-Controller and SDN-Switches for topology by restricting the sending discovery packets to only SDN-Switch ports connected to switches and not to hosts.	Controller	Periodic	(1) Reducing CPU overhead. (2) Reducing the bandwidth overhead.	Port classification consumes more messages
SLDP	A new packet format was used for topology discovery messages with a random source MAC address.	Controller	Periodic	(1) Reducing CPU overhead. (2) Reducing the bandwidth overhead.	Increases Flow_Mod messages to enable authorized packet forwarding
TEDP-S	Reducing messages between the SDN-Controller and SDN-Switches by sending only one discovery packet to the root SDN-Switch.	Controller and Switches	Periodic	(1) Reducing CPU overhead. (2) Reducing the bandwidth overhead.	Increasing CPU overhead on the switches.
TEDP-H	Offloading the process of discovering the topology from SDN-Controller to the root SDN-Switch.	Controller and Switches	Periodic	(1) Reducing CPU overhead. (2) Reducing the bandwidth overhead.	Increasing CPU overhead on the SDN-Switches.
SDN-RDP	Sharing network state management between multiple SDN-Controllers.	Controller	Periodic	(1) Reducing the number of messages. (2) Reducing the computation time.	Manual configurations
GTOP	Improve topology discovery process in PCE to be as OpenFlow	PCE and Switches	Periodic	(1) Reduce link failures (2) Reduce updating times	Legacy domain
SONT	Test-signal mechanism to detect network links	Controller and Optical switches	Periodic	(1) Reduce updating times	Fault tolerance is not checked despite its importance in optical networks
HDDP	A lightweight agent and network exploration model based on flooding	Controller and Switches	Periodic	(1) Support different type of networks	CPU Overhead Increasing packet messages
eTDP	Distributed topology discovery process on layer 2 and uses shortest control paths	Switches	Periodic	(1) Reduce discovery time and cost	Back to traditional networks
TDP	Rely on network partitioning and using a timer to send topology discovery packets	Wireless nodes	Periodic	(1) Reducing send packets (2) Reduce topology discovery energy	Suitable for tree network topology only

Based on Figure 8, the SDN-Controller will send one LLDP Packet-Out for each SDN-Switch in the network. S1 will receive a Packet_Out message and extract an LLDP packet from it. Then it will forward the LLDP packet to all active ports in the SDN-Switch and replaces the source MAC address of the LLDP packet with the MAC address of the

egress port. S2 will receive the LLDP packet and encapsulate the LLDP packet with a Packet_In message and send it to the SDN-Controller. The SDN-Controller will parse the incoming Packet_In packet and learn about the new link. Finally, the results show that OFDPv2 uses 63–80% fewer LLDP Packet_Out messages than the same process in standard OFDP. Furthermore, in measuring the CPU overhead between OFDP and OFDPv2, the results show that OFDPv2 reduced the CPU utilization of the SDN-Controller by up to 45% compared to the standard OFDP.

Additionally, in [42], the authors presented a lightweight, efficient, and secure approach to discover the links between SDN-Switches in SDN called Secure and Lightweight Link Discovery Protocol (SLDP). In general, the proposal uses a new packet format for link discovery by using minimal features of the frame and removing unnecessary features from the standard LLDP frame. Moreover, for each iteration of the link discovery process, the SDN-Controller will generate an SLDP packet and send it with a random source MAC address to SDN-Switches. Then the SDN-Controller installs a flow entry in each SDN-Switch flow table to generate the packet with that random source and agree to these values when a message is returned to the SDN-Controller. In brief, the SDN-Controller will initially send an SLDP packet to each SDN-Switch port in the network, and the subsequent discovery iterations will only receive ports that are eligible for SLDP packets. As result, only legitimate SLDP packets will be sent to the SDN-Controller to build the topology and this will reduce the number of packets used in the discovery process and prevent unqualified ports from receiving SLDP packets. In evaluation metrics, the authors used the Mininet emulator and compared it to the OFDP in various network topologies with different numbers of SDN-Switches, hosts, and links. Moreover, they used the number of packets sent by the SDN-Controller, CPU of the SDN-Controller, and validation time. For all of these metrics, SLDP outperforms the standard OFDP.

Moreover, [43], the authors proposed another approach called Efficient and Secure Link discovery scheme (ESLD) by also limiting the transmission of LLDP packets to SDN-Switch ports connected to SDN-Switches. This approach is illustrated in Figure 9.

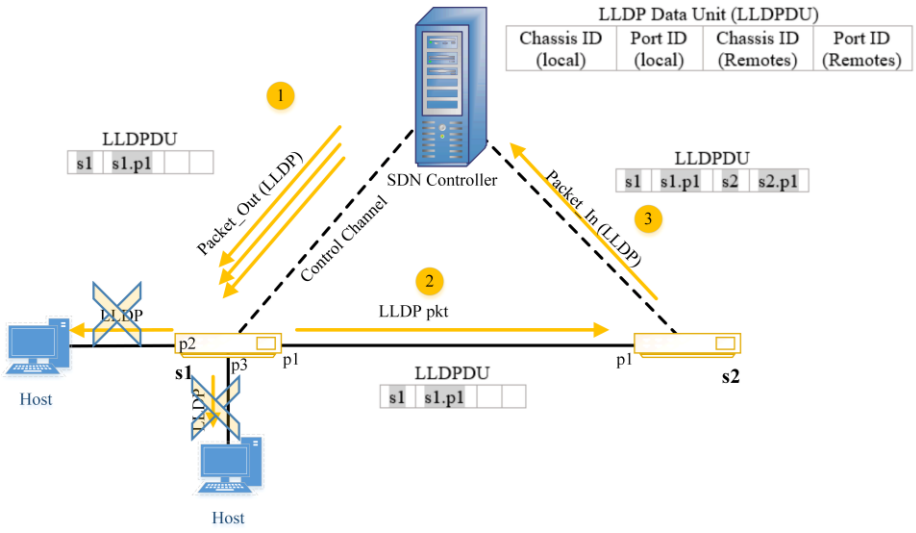

Figure 9. ESLD methodology.

The basic idea of the ESLD is to classify SDN-Switch ports into two classes, either 'Switch' or 'Host'. 'Switch' ports are those ports connected to SDN-Switches and 'host' ports connected to users. Moreover, ESLD uses some well-known OpenFlow messages such as 'Feature-Replay', 'State-Reply', and 'Port-Status' messages to label these ports in both types ('Host' or 'Switch'). Therefore, the efficiency of ESLD is directly dependent on the number of SDN-Switch ports in SDN. For evaluation, the authors used a host scale in different scenarios with different topologies and compared ESLD to the de facto OFDP

and OFDPv2. Moreover, a number of LLDP packets handled by the SDN-Controller, CPU utilization for SDN-Controller, and SDN-Switches were used as performance metrics.

In a different study, the authors suggested two new applications of the OFDP to improve the link discovery process: the Enhanced Topology Discovery Service (ETDP-SDN) and the ETDP-Hybrid [45]. In ETDP-SDN, the Discovery Service is centralized inside the SDN-Controller, and the SDN-Controller identifies each iteration of the root SDN-Switch and then sends an ETDP discovery packet to the root SDN-Switch. SDN-Switch will in turn flood this packet to all of its ports except for the port where ETDP is received. Next, the SDN-Controller installs a flow rule to force SDN-Switch to send a Packet_In to the SDN-Controller and another flow rule to flood the ETDP frame. Next, after the SDN-Controller receives the Packet_In message, it will send an additional FLOW_Mod to remove the flood rule, which was previously installed to prevent loops. However, the ODFP performance is improved by sending a single ETDP packet encapsulated with a Packet_Out message to the SDN-Switch root from the SDN-Controller. This will reduce Packet_Out messages per discovery frequency to just one packet. ETDP also provides the minimum latency path between any two SDN-Switches.

In the second suggested application (ETDP-Hybrid), the discovery service is shared between the SDN-Controller and SDN-Switches. SDN-Switches in this application do not depend on the SDN-Controller to start the discovery process, it initiates the discovery process and installs a flow entry in its flow table to send the topology information to the SDN-Controller. In discussing the results, the authors evaluated their application using Mininet in different topologies and different experimental times to compute the mean and standard deviation. The Packet_out count, Packet_In count, and Flow_Mod count messages were used as performance metrics. For ETDP-SDN, the results showed that only one Packet_out message per SDN-Switch. The number of flow_Mod messages was expected to be twice the number of flow_mod messages in standard OFDP, but that was not the case because the locking mechanism (to stop flooding frames and prevent loops) is not fast enough to stop annoying ETDP packets before installing block flow rules. In ETDP-Hybrid, there were no Packet_Out and Flow_Mod messages sent by the SDN-Controller.

Similarly, the authors presented the SDN Resource Discovery Protocol (SDN-RDP) in [82] as a solution to sharing network state management between multiple SDN-Controllers. Each SDN-Controller discovers a portion of the network topology to maintain distributed node management and enhance protocol accuracy. The presented approach is asynchronous, does not require full network information, and does not need a global startup step. According to the simulated results, the proposed method effectively reduces the overload of the SDN-Controller.

Other topology discovery methods have used centralized techniques, such as the Path Compute Element (PCE). The authors in [83] suggested a centralized topology discovery algorithm called Generalized TOPolgy (GTOP) for PCE. GTOP works by enabling PCE to mechanically create a network topology without employing a global routing protocol such as the Open Short Path protocol (OSPF). The GTOP uses an out-band control channel to proactively collect topology costs from the SDN-Switches. Additionally, it uses the same control channel to update the topology changes in an interactive way. However, concerning the testbed system, the total time for the suggested protocol was 10 ms to update the topology changes. The drawback of this work was using an out-band control channel, which may not be feasible to deploy in large-scale states.

In discovering SDN-Optical Network Topology (SONT), the authors in [41] adopted a sequence of signal checking to detect the links one by one. This technique fits into the pre-service category of layer 1 convergence discovery cases and provides a correct links mapping at SDN-Controller despite limiting scalability and time efficiency, especially in large networks. Therefore, the same authors [84] used a parallel mode rather than a sequential mode to overcome their limitations.

Likewise, the authors in [85] proposed Hybrid Domain Discovery Protocol (HDDP) to enhance the mechanism of discovery topology between SDN and traditional networks. HDDP is managed by SDN-Controller and works as OFDPv2 through the use of a lightweight agent that implements HDDP and transmits topology information indirectly to the SDN-Controller. HDDP uses a network exploration model based on a controlled flooding mechanism (Packert_Out) to discover non-SDN devices. Subsequently, the authors in [40] developed the HDDP to cover the topology discovery process in different types of wireless networks.

Furthermore, other proposed work transfers the topology discovery process from SDN-Controller to layer-2 (SDN-Switch) completely as described in [66]. The authors proposed a new method called enhanced Topology Discovery Protocol (eTDP) which has been classified the SDN-Switches into three types (core, leaf, and v-leaf). Leafs called for SDN-Switches that have only one adjacent switch while SDN-Switches that have more adjacent Switches called v-leaf. The remaining SDN-Switches are called a core. In addition, SDN-Switch ports also have different states according to their positions in the control tree formed by the topology discovery module (Standby, Parent, Child, and Pruned). These performed functions of ports and SDN-Switches are rotated between managed components and independent managers. Each performed function has four tasks: monitor, analyze, plan, and execute. Finally, the SDN-Controller will draw the network topology from the shortest control paths transmit to it by the SDN-Switch core.

Moreover, other works related to topology discovery processes in low power and CPU devices have been discussed in [67,86]. The authors of [67] analyzed the effect of OpenFlow performance over wireless networks in terms of QoS metrics. Meanwhile, the authors in [86] proposed a new topology discovery method called efficient Topology Discovery Protocol (TDP) to achieve simultaneous adaptive SDN-Node ID mapping and topology discovery for Underwater Acoustic Networks (UAN). To save power, the proposal eliminated outdated information about the metering network and node identification from the network topology procedure, leaving only the node identifier. Instead, the SDN-Controller separates the network into layers and employs a local timer in each SDN-Node to ensure that topology discovery packets are sent on time. Each node could determine the network topology and create its SDN-Node ID separately by utilizing the received topology discovery packets.

3.1.2. Event

This domain is named by event, because the changes come along each discovery takes place on every event, such as updating, adding, and deleting links. The authors in [38,76,87] used this type as we will discuss.

In different techniques related to improving OpenFlow topology discovery called Secure and Efficient OpenFlow Topology Discovery Protocol (sOFTDP) [38]. The authors in [38] have transferred the control logic for topology discovery and its security from SDN-Controller to SDN-Switch in order to reduce arithmetic operations from SDN-Controller. SDN-Switches are now responsible for monitoring the status of their ports, and they have used Bidirectional Forwarding Detection (BFD) protocol [88] as a mechanism for detecting port vitality. Each SDN-Switch will establish BFD sessions with their neighbors using three-way handshaking. The SDN-Switches then will use the established BFD session to exchange control and echo messages to monitor link vitality. When a link failure occurs, the affected SDN-Switches will detect this failure by BFD and immediately send a notification called BFD-Status message to the SDN-Controller which also immediately removes this link from its links table. For a link addition, asynchronous notifications using the OFPT__Port_Status message will be sent to the SDN-Controller which in turn will discover this link by following the same procedure that is used in OFDP but only for this link. Moreover, port-status messages (OFPT_Port_Status) are sent from the SDN-Switch to the SDN-Controller when a change in port status occurs or when a new port is added, removed, or modified in the SDN-Switch data path.

The experiments were performed using learning time and CPU utilization performance metrics to evaluate sOFTDP. In link addition, the average learning time for 50 experiments was 5.68 ms while in link removal the learning time was 3.25 ms. Also, they evaluated sOFTDP against standard OFDP and OFDPv2 in terms of CPU utilization of the SDN-Controller throughout 200 s with one topological change in second. The results show that sOFTDP induced the least overhead versus standard OFDP and OFDPv2.

In the same procedural, the approach called ForCES based optimal network topology discovery [76] was proposed. In ForCES, the authors improved OFDP performance by offloading the SDN-Controller from sending or receiving LLDP advertisements. The LLDP protocol will work as it is and without any modifications. The SDN-Switches transmit LLDP advertisements and build their topology tables without the intervention of the SDN-Controller. Then the SDN-Controller pulls the topology information from the SDN-Switches continuously. This technique has been used in [87] in a small-scale testbed. As a result, this proposal can be beneficial by reducing the learning time, and the affected SDN-Switches will inform the SDN-Controller of this change immediately when a change to the topology occurs. However, this solution has been suggested for SDN networks that use ForCES protocol as a southbound interface but can be used with the logic for OpenFlow networks. Finally, the average time for the SDN-Controller to learn about topology changes is 10 ms which is less than 90% of the learning time in standard OFDP (i.e., 100 ms).

3.2. Flow Table Management Algorithms

As we mentioned earlier in the section of switches discovery in SDN technology, the adoption of switch discovery depends mainly on the data and instructions of those flow tables. Each OpenFlow in each switch contains at least one flow table and a set of flow entries within that table. These flow entries contain matching fields, counters, and instructions to apply to matched packages. Typically, it will have more than one flow table, so it is important to note that matching starts from the first flow table and may continue with additional pipeline flow tables. The packet will first start in Table 5 and check these entries based on priority. It will match the highest priority first (e.g., 200, then 100, then 1). If the stream needs to continue to another table, the go-to instruction tells the package to go to the table specified in the instruction. Therefore, improving the technique of dealing with tables will improve the performance of the entire network. The researchers attempted to improve the mechanism of flow tables in the process of topology discovery. Whereas SD-OpenFlow uses TCAM memory to store flow tables, it is very expensive and has a limited size. Thus, the number of flow entries that can be accommodated is limited, therefore, the researchers tried to optimize the flow schedule to take full advantage of it. Moreover, a comparison of these proposals is provided in Table 5 as well.

Table 5. Comparison of flow table management to improve OpenFlow performance.

Proposal	Methodology	Controller Placement Mode	Operation Mode	Goals
Rifai et al. [89]	Flow entry compression	Reactive	Traffic engineering	Maximize the utility of flow tables
Panda et al. [90]	Dynamic hard timeout allocation	Reactive	LRU	Maintain unpredictable flow for a limited period
Isyaku et al. [91]	Dynamic idle and hard timeout based on traffic pattern to reduce overhead	Reactive and Proactive	LRU	Improved the restricted flow table

Table 5. *Cont.*

Proposal	Methodology	Controller Placement Mode	Operation Mode	Goals
Xu et al. [92]	merging flow table and cost of the SDN-Controller	Reactive	Traffic engineering	Adjusts the idle timeout value based on the flow
Kotani and Okabe [93]	packet filtering scheme	Proactive	Traffic engineering	Reducing multiple packet-in messages forwarded to the SDN-Controller
Favaro and Ribeiro [94]	Blackhole mechanism	Reactive	Flow-table management	Maintain visibility for each new flow.
Leng et al. [95]	Rule optimization and binary tree aggregation	Reactive	Flow-table management	Reduce the number of flow entries
Li et al. [96]	Used Q-Learning rule for selecting effective timeout values	Proactive	Machine Learning	Adjusts the idle timeout value based on the flow
Yang and Riley [97]	Classify flows into active and inactive to decide the right flow to remove intelligently	Proactive	Machine learning	Increase the flow table capacity

Timeout mechanism is one of the techniques used by the SDN-Controller to calculate the lifespan of the forwarding entry in the switch flow table. When no packet matches an item during its timeout period, the entry is ejected from the flow table, making room for fresh arriving packets [89]. There are now two methods for installing the timeout mechanism in the OpenFlow SDN-Controller: idle and hard timeout. Typically, the OpenFlow SDN-Controller configures each stream's flow entry with a preset idle timeout value in seconds. However, when it comes to packet inter-arrival time, the flow table has enough room to accommodate all flows, such a timeout number may provide superior performance. In [90], the authors consider a dynamic allocation of hard timeout for identified and unidentified flows. The goal is to maintain unidentified flow for a limited period while also maintain the identified flows. This is accomplished by studying packet traces to determine the nature of each packet's arrival and then modifying the hard timeout accordingly. When a flow table's load is exceeded, expires Least Recently Used (LRU) is utilized to remove an entry with a maximum timeout. Even when the strategy has succeeded in maintaining identified flows, the challenge of a bad entrance removal scheme remains. LRU is not an ideal eviction method for SDN since it is a packet-driven method that can only be executed in theory but may not be suitable with OpenFlow in action. In the same dynamic allocation technique, the authors in [91] used dynamic inertia value for short live flows and set the hard timeout value for long live flows with a short time between packets arrival. This has greatly reduced the Packet_In numbers and as a result, reduced SDN-Controller overhead. By utilizing OpenFlow's built-in data collecting, the limitation of LRU is resolved. Flows with a low packet count are victims, so deleting them improves the amount of the limited storage. Without modifying the design of SDN, it is effective to conclude that their proposal reduced network cost and improved the restricted flow table usage to some level. In [92], the authors offered an adaptive flow table alteration based on the fraction of active flow entries by merging the flow table and cost of the SDN-Controller. The algorithm constantly examines traffic, and as a result, the procedure adjusts the idle timeout value based on the flow. Surprisingly, the algorithm was able to set different timeout settings for distinct flows. On the other hand, the process of determining the cost of flow table entries in the cost of the switch and SDN-Controller computing adds additional computational expenses to the SDN-Controller.

Another technique that was used to reduce SDN-Controller overhead, the authors in [93] employed a packet filtering scheme. The mechanism started by inspecting the header in Packet-In messages and dropping the duplicate one. Moreover, in [94] the authors devised a blackhole technique to reduce SDN-Controller overhead and preserve the advantage of visibility for each new flow. The packet forwarding architecture between SDN-Switch and SDN-Controller has been changed so that only the first packet is routed to the SDN-Controller and successive table-misses are dealt with locally. In addition, its architectural design leaves it vulnerable to dropping alerted packets, which could result in a substantial number of packet losses. However, the amount of events transmitted to the SDN-Controller is minimized. In addition, another technique based on flow entries aggregation has been proposed in [95] to reduce the load on TCAM. Theoretically, the strategy reduces the number of entries to be saved by compressing fine-grained sending entries into less coarse-grained entries with a somewhat greater matching range. Surprisingly, it is a software application that is easy to install as an additional plug-in on the OpenFlow SDN-Controller and does not require additional hardware. However, since the aggregation methodology fails to preserve the original semantics of the rule in most circumstances, the SDN-Controller has certain issues when changing the direction of entries or querying the traffic stats counter.

A different technique based on machine learning has been proposed to identify the classes of traffic flows [96,97] to predict the duration of the flow entry. In [97], the authors used machine learning techniques to determine which flow should be eliminated. The algorithm, which is based on past data of flow entries, forecasts and with the flow entry with the shortest time being the victim. Depending on the output of the algorithm used, it is determined whether the flow entries are active or inactive. Moreover, in [96], the authors also used the Q-Learning rule for selecting effective timeout values for flows to improve OpenFlow SDN-Switch performance. These techniques, on the other hand, may provide superior performance in a small–medium-sized network, but a large-scale dynamic network may necessitate a more complex training set, which in turn necessitates greater storage to handle more historical data.

3.3. Control Channel Improvement Algorithms

Furthermore, in Section 2, we mentioned the importance and how the OpenFlow protocol control channel works. Where it is the hot line of communication between the controller and the plane layer devices. There are a number of constraints for both out-of-band and in-band controls. On the one hand, out-of-band control is costly and inconvenient. This is due to the fact that each data plane device requires additional physical interfaces and cabling to link to the controller. Furthermore, data plane devices may be physically positioned far from the controller, necessitating a significant additional cost to construct a network protocol for out-of-band control. In this subsection, we will discuss the proposed solutions that are related to control channel failure detection and recovery, where sustaining communication between the SDN-Controller and SDN-Switches becomes the main difficulty during network disruptions. Moreover, a comparison of these proposals is provided in Table 6 as well.

As the control channel is used to establish in-band and out-band connection between the SDN-Controller layer and data layer in OpenFlow-SDN, the control channel failure threatens the network availability. According to [98,99] a lossy control channel dramatically reduces data layer throughput and response time. Therefore, several recent works have been proposed to help sustain OpenFlow availability. The authors in [100] proposed control channel recovery for in-band and out-band control links. To enable local recovery from failure, the author integrated logical ring topology with source-routed forwarding. The SDN-Switch-to-SDN-Controller communication was made more robust using a ring-based local recovery technique. A source forwarding method was employed to ensure the SDN-Controller-to-SDN-Switch communication was robust, as a robust SDN-Switch-to-SDN-Controller communication channel allows the SDN-Controller to be updated of

the complete topology. However, while keeping the functionality of data plane devices, this recovery does not necessitate SDN-Controller involvement. Moreover, [98] protected the in-band control channel from failure by finding a set of ideal pathways. To recover from SDN-Switch failure in an SDN, this research used a bypass and non-intervention backup recovery methodology. When an SDN-Switch fails, the goal is to deal with it as quickly as possible, especially if it is part of an SDN-Switch group. If a group of dependent SDN-Switches is affected by the failure, all control traffic from the impacted SDN-Switches is re-routed to a different recovery path.

Table 6. Comparison of control channel schemes to improve OpenFlow performance.

Proposal	Methodology	Controller Placement Mode	Operation Mode	Logic Location
Asadujjaman et al. [100]	Combined between topology type and source-routed forwarding to support local failure recovery	In-band	Recovery	SDN-Switch
Fan and Yang [99]	Centralized trust management system for in-band control channel	In-band	Protection and Recovery	SDN-Controller
Osman et al. [101]	The hybrid controlling mode that dynamically changes between centralized and distributed	In-band and Out-band	Protection	SDN-Controller and SDN-Switches
Alowa and Fevens [98]	Trusted control pathways for in-band control channel	In-band	Protection and Recovery	SDN-Switches
Hwang and Tang [102]	weighted function (Complete Bipartite Graph) technique is used to select the alternative control channel path	In-band	Recovery and Protection	SDN-Switches
Ko et al. [103]	Dijkstra algorithm is used to calculate the shortest control channel pathways	In-band	Protection and Recovery	SD-Swatches
Chan et al. [60]	K-best is used to find control channel paths in between multiple controllers	In-band	Protection	SDN-Controller and SDN-Switches
Ibrar et al. [104]	Logistic regression and support vector machine algorithms to predict the link status	In-band	Protection	SDN-Controller
Yang and Riley [97]	Classify flows into active and inactive to decide the right flow to remove intelligently	Proactive	Machine learning	Increase the flow table capacity

Using a full cartelized control channel, the authors in [99] suggested a centralized trust management system for selecting the most trustworthy data transfer pathways (in-band control) in the SDN. The authors viewed the network as a multi-agent system, with SDN-Switches and routers serving as the agents. The authors thought that their centralized unit had complete control over the route choices made by the agents. Fully centralized systems, in most circumstances, necessitate highly complex resources for the central unit, while also introducing a single point of failure. The central unit must be completely operational, tamper-resistant, and physically secure. It is also necessary to keep a close eye on all of the system's agents. The cost of applying such systems rises dramatically when all of the above-described qualities are combined. In contrast, the authors in [101] proposed a new method called hybrid-SDN to operate the network regardless of the level of control vulnerability by

shifting network control from central to distributed. This approach depends on the Control Packet Loss Ratio (CPLR). The OpenFlow-SDN is converted to a hybrid-SDN (distributed control channel to SDN-Switches) approach when the CPLR value reaches an undesirable level.

In terms of restoring the OpenFlow control channel failover mechanism, SDN-Switches can detect a broken link but must wait for the SDN-Controller to generate alternate pathways. Therefore, the authors in [102,103] looked at how to protect and recover from control channel failures. Authors in [102] employed both recovery and protection methods for data channel failures. In the recovery process, the SDN-Controller constructs alternative paths depending on the complete two-segment graphs of the topology of the network, and the SDN-Controller will use the weighted functions to the alternative paths to rebuilt SDN-Switches control channel path. Moreover, in control channel protection, the SDN-Controller identifies flow entries for the SDN-Switches. However, this proposal has drawbacks related to both processes (recovery and protection) as the increase of failover times as the number of SDN-Switches grows and handling with rapid changes in network topologies. Therefore, these drawbacks were overcome in [103] by using both a flexible network hypervisor-based structure. The suggested solution involves calculating backup pathways (Dijkstra algorithm) in a finer timeframe (e.g., 5 s), and only configuring recovery flow rules if a physical network failure is detected. In addition, another recovery control channel based on the use of multiple SDN-Controllers in in-band networks has been proposed in [60]. The master SDN-Controller is in charge of network control at the stable level, while the other SDN-Controllers are on standby to take over the network control in the event of failure. The authors planned the control paths using the K-best path technique on a modified graph, and in this way, each SDN-Switch would be serviced by several SDN-Controllers with discontinuous pathways.

Furthermore, the authors in [104], attempted to reduce the impact of control channel service failure by predicting link failure before it occurred. The authors used a machine learning technique for their proposed method that re-computes the locations of access control policies and reduces their violation in the event of link failure.

4. Discussion and Open Issues

Each of the proposals presented in the previous section has its characteristics of improving one or more parts of the topology discovery process. All of these approaches that seek to improve the process of discovering different or similar topologies must be aligned with the basic criteria of OpenFlow-SDN protocol, as we will discuss below.

4.1. Location of the Topology Discovery Logic

In these OFDP, OFDPv2, SDN-RDP, ESLD, and SLDP protocols, the control logic for the topology discovery remained in the SDN-Controller. In OFDPv2, the SDN-Controller starts the discovery process and sends LLDP advertisements to the SDN-Switches. The SDN-Switches have a secondary role in the discovery process; It just consists of forwarding the SDN-Controller's LLDP advertisements to its neighbor, or encapsulating the LLDP advertisement in a Packet_In message and send it back to the SDN-Controller. Whereas in TEDP-H and sOFTDP protocols, almost all of the logic is in SDN-Switches. The SDN-Switch is responsible for monitoring the status of links with its neighbors and for sending notifications when there is a change in the topology. The SDN-Controller plays a secondary role and is only responsible for sending an LLDP advertisement when it receives an OFPT_Port_Status message indicating that the SDN-Switch port state is changing. In eTDP and ForCES, the logic of topology discovery is entirely in the SDN-Switches. SDN-Switch is responsible for creating and forwarding LLDP advertisements independently and the SDN-Controller only captures link information from the SDN-Switches. In HDDP and TEDP-S, the logic is distributed between the SDN-Switches and the SDN-Controller.

4.2. How Much Do Methods Differ from OFDP

This standard discusses the number of modifications of each proposal to standard OFDP. OFDP takes advantage of the OpenFlow channel to exchange messages between the SDN-Controller and SDN-Switches. HDDP, SDN-RDP, and OFDPv2 make minor modifications to OFDP, as they still use OpenFlow messaging. In sOFTDP, the topology discovery depends entirely on the BFD protocol to detect link failures, which is considered a major change in the methodology of OFDP. In addition, eTDP and sOFTDP rely on a new notification message (i.e., BFD-Status) that is not defined in OpenFlow specification. Moreover, the eTDP uses a new technique that depends on SDN-Switch classes. In ForCES, it is just an implementation of the LLDP protocol for ForCES-based SDNs. The methodology of the discovery process is very different from that of OFDP and requires modification of OFDP to a large extent. In ESLD and SLDP, the topology discovery methodology is the same as in OFDP except for some other steps to limit the transmission of discovery packets to SDN-Switch ports that connect to other SDN-Switches. In TEDP, the methodology is very different from OFDP. The SDN-Controller sends a Packet_out message to each active SDN-Switch port in standard OFDP, but in TEDP, the SDN-Controller sends only one Packet_out to the root SDN-Switch, which in turn completes the task.

4.3. Operation Methods

By operation method, we mean that if the discovery of the topology is performed in a periodic mode or is triggered by events. 'Periodic' meaning that the SDN-Controller inquires about the network topology within a specified period, while the 'triggered' type means that the process of changing the network topology takes place only when changes occur in network topology. However, most of the reviewed works that we reviewed were using the periodic method and two of them only used the triggered. OFDPv2, ESLD, SLDP, and TEDP periodically discover the topology as in standard OFDP. Whereas in sOFTDP, the topology discovery is triggered by topology change events. In ForCES, the LLDP advertisements are periodically sent between SDN-Switches and if there is a change in the topology, the affected SDN-Switches will notify the SDN-Controller about this change.

However, each type of operation method has its pros and cons. This opens the door to the field of application of a hybrid model between the two methods, which can take advantage of the advantages of each type.

4.4. CPU Usage

While OFDPv2 reduces the number of LLDP Packet_Out messages from the number of active SDN-Switch ports in the network to only the number of SDN-Switches. Results in reducing the CPU usage of the SDN-Controller by up to 45%. In eTDP and sOFTDP, the discovery process runs without the need for frequent advertisements. This significantly reduces the number of LLDP Packet_Out and Packet_In messages and then reduces CPU usage. However, sOFTDP requires SDN SDN-Switches to support a completely independent protocol (i.e., BFD protocol) beside OpenFlow protocol. In ForCES, the SDN-Controller is offloaded from sending LLDP advertisements, which reduces CPU usage drastically. In addition, the learning time for topology changes, particularly link failures are still relatively high because the SDN-Controller will capture the information about topology change after notifications from the affected SDN-Switches. Moreover, CPU utilization in SDN-Switches is increased ESLD, HDDP, and eDTP due to the transformation of the process of topology discovery to the SDN-Switches. In SLDP, the results showed significantly lower CPU usage compared to OFDP. In TEDP, Packet_Out messages sent from the SDN-Controller are distributed to SDN-Switches which in turn reduce the CPU usage. In HDDP, also the SDN-Controller will be overhead due to the flooding Packet-Out messages. TDP and eTDP their technique is offloaded from SDN-Controller to SDN-Switches. Moreover, in [105], the functionalities were offloaded from the data plane to the smart network interface.

4.5. Learning Time

In OFDPv2, ESLD and SLDP learning time is still equal to OFDP, which is high. In SDN-RDP, eTDP, and sOFTDP, learning time is less than OFDP and OFDPv2. In ForCES, the learning time for topology changes, particularly link failures is still relatively high because the SDN-Controller will capture the information about topology change after notifications from the affected SDN-Switches. In HDDP and TEDP, the learning time is the largest because the discovery packet will traverse the whole of the network before discovering the topology change.

However, the process of topological discovery in SDN networks is one of the essential principles that must be given priority in terms of performance and security. The three main components (i.e., Link, Control Channel, and Flow-Table) of the topology discovery process must ideally work together or close to ideal to achieve optimal performance. Thus, optimizing the control channel communication in both methods (protection and recovery) was discussed in the above section but still needs to be covered based on the two redundant paths (in-band and out-band). Moreover, the recent references to OpenFlow defined by ONF [58] already define the architecture of an OpenFlow generically, in a client-server manner, rather than in a data-control plane. Therefore, in our opinion, the future directions of topology discovery should focus on offloading most of the data layer functionalities to programmable hardware such as smart network interfaces are a good solution. Moreover, heterogeneous networks such as wire and various types of wireless also should be included in the topology discovery process, to become more realistic in real environments. In addition, the use of machine learning techniques with programmable devices at the data level to reduce the incidence of errors in topology discovery to rates close to zero. Furthermore, most of the studies reviewed in topology discovery focused on periodic operational mode and ignoring the event. Even the event operational mode will provide a good solution to reduce the process's cost. Moreover, the combination between both operation modes will take into account another good solution to improve the topology discovery process.

5. Conclusions

While the SDN architecture appears to solve problems within the traditional network architecture, it also comes with some major challenges. In this paper, we highlight one of these challenges, which is related to the topology discovery service. In standard SDN, the SDN-Controller is responsible for maintaining an updated network topology through using OFDP protocol to discover the links between SDN-Switches in the data plane. However, it has major limitations in its performance, especially in huge and dynamic networks. Moreover, applications at the application layer depend entirely on that topology. Therefore, several limitations related to the performance of the link discovery protocol used in SDN networks (OFDP) are presented in this paper. Furthermore, the elements (flow tables and control channels) related to the OpenFlow protocol and optimization studies on these elements are explained, which in turn leads to improving the performance of OFDP. However, there are not a large number of studies that attempt to address these limitations in the literature. Therefore, it is a good opportunity for researchers to work on it for their future studies. Moreover, five fundamentals that measure the similarity between the proposed technique and the standard (OFDP) used in the topology discovery are discussed, including topology discovery logic location, operation mode, OFDP compatibility, CPU usage, and learning time. Based on these fundamentals, each of these proposals has some limitations in one or more of these fundamentals. Therefore, we still need a new proposal that takes into account all five fundamentals. One of the research directions in the development of a new topology discovery approach that distributes the logic between SDN-Switches and SDN-Controller. This solution will take as little workload as possible on both and reduce learning time as much as possible. As another research direction for proposals that discover the topology periodically, the trade-off between SDN-Controller overhead and learning time to meet the requirements of the environment must be determined. This

trade-off should be based on the criticality of the environment and topology change rate. In some environments, such as enterprise networks, the rate of topology change is low and less significant; thus, it is possible to increase the discovery interval. In contrast, in data-centers and transport networks, the rate of topology change is high and critical, thus, it is recommended to decrease the discovery interval as much as possible.

Author Contributions: All authors contributed to this manuscript. Conceptualization, R.A., R.W. and S.A.; Investigation, R.A., S.A. and R.W.; Data duration, R.A., R.W. and S.A.; Writing—original draft, R.A., S.A. and R.W.; Visualization, R.A.; Supervision, R.A.; Writing—review and editing, R.A., R.W. and S.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Feamster, N.; Rexford, J.; Zegura, E. The road to SDN: An intellectual history of programmable networks. *Comput. Commun. Rev.* **2014**, *44*, 87–98. [CrossRef]
2. Kreutz, D.; Ramos, F.M.V.; Verissimo, P. Towards secure and dependable software-defined networks. In Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, Chicago, IL, USA, 22 August 2014; pp. 55–60. [CrossRef]
3. Dargahi, T.; Caponi, A.; Ambrosin, M.; Bianchi, G.; Conti, M. A Survey on the Security of Stateful SDN Data Planes. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1701–1725. [CrossRef]
4. Kirkpatrick, K. Software-defined networking. *Commun. ACM* **2013**, *56*, 16–19. [CrossRef]
5. Sezer, S.; Scott-Hayward, S.; Chouhan, P.K.; Fraser, B.; Lake, D.; Finnegan, J.; Viljoen, N.; Miller, M.; Rao, N. Are we ready for SDN? Implementation challenges for software-defined networks. *IEEE Commun. Mag.* **2013**, *51*, 36–43. [CrossRef]
6. Levin, D.; Wundsam, A.; Heller, B.; Handigol, N.; Feldmann, A. Logically centralized? In Proceedings of the First Workshop on Hot Topics in Software Defined Networks—HotSDN '12, Helsinki, Finland, 13 August 2012; ACM Press: New York, NY, USA, 2012; p. 1. [CrossRef]
7. ONF SDN Architecture ONF. Available online: https://www.opennetworking.org/sdn-definition (accessed on 8 June 2021).
8. Hakiri, A.; Gokhale, A.; Berthou, P.; Schmidt, D.C.; Gayraud, T. Software-Defined Networking: Challenges and research opportunities for Future Internet. *Comput. Netw.* **2014**, *75*, 453–471. [CrossRef]
9. Kim, H.; Feamster, N. Improving network management with software defined networking. *IEEE Commun. Mag.* **2013**, *51*, 114–119. [CrossRef]
10. Shirali-Shahreza, S.; Ganjali, Y. Efficient Implementation of Security Applications in OpenFlow Controller with FleXam. In Proceedings of the 2013 IEEE 21st Annual Symposium on High-Performance Interconnects, San Jose, CA, USA, 21–23 August 2013; pp. 49–54. [CrossRef]
11. Van Adrichem, N.L.M.; Doerr, C.; Kuipers, F.A. OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks. In Proceedings of the 2014 IEEE Network Operations and Management Symposium (NOMS), Krakow, Poland, 5–9 May 2014; pp. 1–8. [CrossRef]
12. Lim, S.; Ha, J.; Kim, H.; Kim, Y.; Yang, S. A SDN-oriented DDoS blocking scheme for botnet-based attacks. In Proceedings of the 2014 Sixth International Conference on Ubiquitous and Future Networks (ICUFN), Shanghai, China, 8–11 July 2014; pp. 63–68. [CrossRef]
13. Xia, W.; Wen, Y.; Foh, C.H.; Niyato, D.; Xie, H. A Survey on Software-Defined Networking. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 27–51. [CrossRef]
14. Jarraya, Y.; Madi, T.; Debbabi, M. A Survey and a Layered Taxonomy of Software-Defined Networking. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 1955–1980. [CrossRef]
15. Kreutz, D.; Ramos, F.M.V.; Esteves Verissimo, P.; Esteve Rothenberg, C.; Azodolmolky, S.; Uhlig, S. Software-Defined Networking: A Comprehensive Survey. *Proc. IEEE* **2015**, *103*, 14–76. [CrossRef]
16. Jammal, M.; Singh, T.; Shami, A.; Asal, R.; Li, Y. Software defined networking: State of the art and research challenges. *Comput. Netw.* **2014**, *72*, 74–98. [CrossRef]
17. Braun, W.; Menth, M. Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices. *Futur. Internet* **2014**, *6*, 302–336. [CrossRef]
18. Jain, R.; Paul, S. Network virtualization and software defined networking for cloud computing: A survey. *IEEE Commun. Mag.* **2013**, *51*, 24–31. [CrossRef]

19. OpenDaylight. Available online: https://www.opendaylight.org/ (accessed on 8 June 2021).
20. Project Floodlight. Available online: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343517/Supported+Topologies (accessed on 8 June 2021).
21. Gude, N.; Koponen, T.; Pettit, J.; Pfaff, B.; Casado, M.; McKeown, N.; Shenker, S. NOX. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 105–110. [CrossRef]
22. Ryu Controller. Available online: https://github.com/OpenState-SDN/ryu (accessed on 8 June 2021).
23. Wang, X.; Gao, N.; Zhang, L.; Liu, Z.; Wang, L. Novel MITM Attacks on Security Protocols in SDN: A Feasibility Study. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, Vancouver, BC, Canada, 4–6 October 2010*; USENIX Association: Vancouver, BC, Canada, 2016; pp. 455–465. [CrossRef]
24. Erickson, D. The beacon openflow controller. In *HotSDN '13. Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, Hong Kong, China, 16 August 2013*; ACM Press: New York, NY, USA, 2013; p. 13. [CrossRef]
25. Dong, L.; Gopal, R.; Halpern, J. Forwarding and Control. Element Separation (ForCES) Protocol Specification. *RFC* **2010**, *53*, 1–24. [CrossRef]
26. Song, H. Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane. In *HotSDN '13. Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, Hong Kong, China, 16 August 2013*; ACM Press: New York, NY, USA; pp. 127–132. [CrossRef]
27. Bianchi, G.; Bonola, M.; Capone, A.; Cascone, C. OpenState. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 44–51. [CrossRef]
28. ONF. *OpenFlow Switch Specification version 1.3.0*; ONF: Menlo Park, CA, USA, 2012.
29. Nisar, K.; Jimson, E.R.; Hijazi, M.H.A.; Welch, I.; Hassan, R.; Aman, A.H.M.; Sodhro, A.H.; Pirbhulal, S.; Khan, S. A survey on the architecture, application, and security of software defined networking: Challenges and open issues. *Internet Things* **2020**, *12*, 100289. [CrossRef]
30. Nunes, B.A.A.; Mendonca, M.; Nguyen, X.-N.; Obraczka, K.; Turletti, T. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 1617–1634. [CrossRef]
31. Correa Chica, J.C.; Imbachi, J.C.; Botero Vega, J.F. Security in SDN: A comprehensive survey. *J. Netw. Comput. Appl.* **2020**, *159*, 102595. [CrossRef]
32. Wang, X.; Gao, N.; Zhang, L.; Liu, Z.; Wang, L. Novel MITM Attacks on Security Protocols in SDN: A Feasibility Study. In *Information and Communications Security, Proceedings of the 18th International Conference, ICICS 2016, Singapore, 29 November–2 December 2016*; Lam, K.-Y., Chi, C.-H., Qing, S., Eds.; Springer International Publishing: Cham, Germany, 2016; pp. 455–465. ISBN 978-3-319-50011-9. [CrossRef]
33. Khan, S.; Gani, A.; Abdul Wahab, A.W.; Guizani, M.; Khan, M.K. Topology Discovery in Software Defined Networks: Threats, Taxonomy, and State-of-the-Art. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 303–324. [CrossRef]
34. Hong, S.; Xu, L.; Wang, H.; Gu, G. Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures. In Proceedings of the 2015 Network and Distributed System Security Symposium, San Diego, CA, USA, 8–11 February 2015; Internet Society: Reston, VA, USA, 2015. [CrossRef]
35. Dhawan, M.; Poddar, R.; Mahajan, K.; Mann, V. SPHINX: Detecting Security Attacks in Software-Defined Networks. In Proceedings of the 2015 Network and Distributed System Security Symposium, San Diego, CA, USA, 8–11 February 2015; Internet Society: Reston, VA, USA, 2015. [CrossRef]
36. Breitbart, Y.; Garofalakis, M.; Martin, C.; Rastogi, R.; Seshadri, S.; Silberschatz, A. Topology discovery in heterogeneous IP networks. In Proceedings of the IEEE INFOCOM 2000 Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064), TelAviv, Israel, 26–30 March 2000; Volume 1, pp. 265–274. [CrossRef]
37. Pakzad, F.; Portmann, M.; Tan, W.L.; Indulska, J. Efficient topology discovery in software defined networks. In Proceedings of the 2014 8th International Conference on Signal Processing and Communication Systems (ICSPCS), Queensland, Australia, 15–17 December 2014; pp. 1–8. [CrossRef]
38. Azzouni, A.; Boutaba, R.; Trang, N.T.M.; Pujolle, G. sOFTDP: Secure and efficient OpenFlow topology discovery protocol. In Proceedings of the NOMS 2018–2018 IEEE/IFIP Network Operations and Management Symposium, Taipei, Taiwan, 23–27 April 2018; pp. 1–7. [CrossRef]
39. Pakzad, F.; Portmann, M.; Tan, W.L.; Indulska, J. Efficient topology discovery in OpenFlow-based Software Defined Networks. *Comput. Commun.* **2016**, *77*, 52–61. [CrossRef]
40. Martinez-Yelmo, I.; Alvarez-Horcajo, J.; Carral, J.A.; Lopez-Pajares, D. eHDDP: Enhanced Hybrid Domain Discovery Protocol for network topologies with both wired/wireless and SDN/non-SDN devices. *Comput. Netw.* **2021**, *191*. [CrossRef]
41. Montero, R.; Agraz, F.; Pages, A.; Perello, J.; Spadaro, S. Dynamic topology discovery in SDN-enabled Transparent Optical Networks. In Proceedings of the 2017 International Conference on Optical Network Design and Modeling, Budapest, Hungary, 15–18 May 2017; pp. 1–6. [CrossRef]
42. Nehra, A.; Tripathi, M.; Gaur, M.S.; Battula, R.B.; Lal, C. SLDP: A secure and lightweight link discovery protocol for software defined networking. *Comput. Netw.* **2019**, *150*, 102–116. [CrossRef]
43. Zhao, X.; Yao, L.; Wu, G. ESLD: An efficient and secure link discovery scheme for software-defined networking. *Int. J. Commun. Syst.* **2018**, *31*, e3552. [CrossRef]

44. Li, Y.; Cai, Z.-P.; Xu, H. LLMP: Exploiting LLDP for Latency Measurement in Software-Defined Data Center Networks. *J. Comput. Sci. Technol.* **2018**, *33*, 277–285. [CrossRef]
45. Rojas, E.; Alvarez-Horcajo, J.; Martinez-Yelmo, I.; Carral, J.A.; Arco, J.M. TEDP: An Enhanced Topology Discovery Service for Software-Defined Networking. *IEEE Commun. Lett.* **2018**, *22*, 1540–1543. [CrossRef]
46. Chen, C.C.; Chen, Y.R.; Tsai, S.C.; Yang, M.C. Forwarding path discovery with software defined networking. In Proceedings of the 2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS), Seoul, Korea, 27–29 September 2017; pp. 299–302. [CrossRef]
47. Dacier, M.C.; Konig, H.; Cwalinski, R.; Kargl, F.; Dietrich, S. Security Challenges and Opportunities of Software-Defined Networking. *IEEE Secur. Priv.* **2017**, *15*, 96–100. [CrossRef]
48. Rawat, D.B.; Reddy, S.R. Software Defined Networking Architecture, Security and Energy Efficiency: A Survey. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 325–346. [CrossRef]
49. Scott-Hayward, S.; Natarajan, S.; Sezer, S. A Survey of Security in Software Defined Networks. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 623–654. [CrossRef]
50. Shin, S.; Xu, L.; Hong, S.; Gu, G. Enhancing Network Security through Software Defined Networking (SDN). In Proceedings of the 2016 25th International Conference on Computer Communication and Networks (ICCCN), Waikoloa, HT, USA, 1–4 August 2016; pp. 1–9. [CrossRef]
51. Akhunzada, A.; Gani, A.; Anuar, N.B.; Abdelaziz, A.; Khan, M.K.; Hayat, A.; Khan, S.U. Secure and dependable software defined networks. *J. Netw. Comput. Appl.* **2016**, *61*, 199–221. [CrossRef]
52. Li, W.; Meng, W.; Kwok, L.F. A survey on OpenFlow-based Software Defined Networks: Security challenges and countermeasures. *J. Netw. Comput. Appl.* **2016**, *68*, 126–139. [CrossRef]
53. Nehra, A.; Tripathi, M.; Gaur, M.S. "Global view" in SDN. In Proceedings of the the 10th International Conference on Security of Information and Networks—SIN '17, Jaipur, India, 13–15 October; ACM Press: New York, NY, USA, 2017; pp. 303–306. [CrossRef]
54. Espinel Sarmiento, D.; Lebre, A.; Nussbaum, L.; Chari, A. Decentralized SDN Control Plane for a Distributed Cloud-Edge Infrastructure: A Survey. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 256–281. [CrossRef]
55. Hayes, M.; Ng, B.; Pekar, A.; Seah, W.K.G. Scalable Architecture for SDN Traffic Classification. *IEEE Syst. J.* **2018**, *12*, 3203–3214. [CrossRef]
56. Alsaeedi, M.; Mohamad, M.M.; Al-Roubaiey, A.A. Toward Adaptive and Scalable OpenFlow-SDN Flow Control: A Survey. *IEEE Access* **2019**, *7*, 107346–107379. [CrossRef]
57. Isyaku, B.; Mohd Zahid, M.S.; Bte Kamat, M.; Abu Bakar, K.; Ghaleb, F.A. Software Defined Networking Flow Table Management of OpenFlow Switches Performance and Security Challenges: A Survey. *Futur. Internet* **2020**, *12*, 147. [CrossRef]
58. ONF. *OpenFlow Switch. Specification 1.5.1*; ONF: Menlo Park, CA, USA, 2015.
59. Narisetty, R.; Dane, L.; Malishevskiy, A.; Gurkan, D.; Bailey, S.; Narayan, S.; Mysore, S. OpenFlow configuration protocol: Implementation for the of management plane. In Proceedings of the 2013 Second GENI Research and Educational Experiment Workshop, Salt Lake City, UT, USA, 20–22 March 2013; pp. 66–67. [CrossRef]
60. Chan, K.Y.; Chen, C.H.; Chen, Y.H.; Tsai, Y.J.; Lee, S.S.W.; Wu, C.S. Fast Failure Recovery for In-Band Controlled Multi-Controller OpenFlow Networks. In Proceedings of the 2018 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Korea, 17–19 October 2018; pp. 396–401. [CrossRef]
61. Tr-510, O.N.F. The Benefits of Multiple Flow Tables and TTPs. Available online: https://www.opennetworking.org/wp-content/uploads/2014/10/TR_Multiple_Flow_Tables_%0Aand_TTPs.pdf (accessed on 2 April 2021).
62. Fancy, C.; Pushpalatha, M. Performance evaluation of SDN controllers POX and floodlight in mininet emulation environment. In Proceedings of the 2017 International Conference on Intelligent Sustainable Systems (ICISS), Palladam, India, 7–8 December 2017; pp. 695–699. [CrossRef]
63. Cisco Cisco Open SDN Controller. Available online: https://www.cisco.com/c/en/us/products/wireless/wireless-lan-controller/index.html (accessed on 8 June 2021).
64. Geni GENI Wiki. Available online: http://groups.geni.net/geni/wiki/OpenFlowDiscoveryProtocol (accessed on 6 June 2021).
65. Mayoral, A.; Vilalta, R.; Muñoz, R.; Casellas, R.; Martínez, R. SDN orchestration architectures and their integration with Cloud Computing applications. *Opt. Switch. Netw.* **2017**, *26*, 2–13. [CrossRef]
66. Ochoa-Aday, L.; Cervello-Pastor, C.; Fernandez-Fernandez, A. ETDP: Enhanced topology discovery protocol for software-defined networks. *IEEE Access* **2019**, *7*, 23471–23487. [CrossRef]
67. Araniti, G.; Cosmas, J.; Iera, A.; Molinaro, A.; Morabito, R.; Orsino, A. OpenFlow over wireless networks: Performance analysis. *IEEE Int. Symp. Broadband Multimed. Syst. Broadcast. BMSB* **2014**. [CrossRef]
68. Aslan, M.; Matrawy, A. On the Impact of Network State Collection on the Performance of SDN Applications. *IEEE Commun. Lett.* **2016**, *20*, 5–8. [CrossRef]
69. Kempf, J.; Bellagamba, E.; Kern, A.; Jocha, D.; Takacs, A.; Skoldstrom, P. Scalable fault management for OpenFlow. In Proceedings of the 2012 IEEE International Conference on Communications (ICC), Ottawa, ON, Canada, 10–15 June 2012; pp. 6606–6610. [CrossRef]
70. Abdelhadi, A.; Boutaba, R.; Pujolle, G. Limitations of OpenFlow Topology Discovery Protocol. In Proceedings of the 16th Annual Mediterranean Ad Hoc Networking Workshop (Med-hoc-Net 2017), Budva, Montenegro, 28–30 June 2017.

71. Tbah, M.; Azzouni, A.; Nguyen, M.T.; Pujolle, G. Topology Discovery Performance Evaluation of OpenDaylight and ONOS Controllers. In Proceedings of the 2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), Paris, France, 19–21 February 2019; pp. 285–291. [CrossRef]
72. Shirmarz, A.; Ghaffari, A. Performance issues and solutions in SDN-based data center: A survey. *J. Supercomput.* **2020**, *76*, 7545–7593. [CrossRef]
73. Bholebawa, I.Z.; Dalal, U.D. Performance analysis of SDN/openflow controllers: POX versus floodlight. *Wirel. Pers. Commun.* **2018**, *98*, 1679–1699. [CrossRef]
74. Mittal, S. Performance Evaluation of Openflow SDN Controllers. *Adv. Intell. Syst. Comput.* **2018**, *736*, 913–923. [CrossRef]
75. Vaghani, R.; Lung, C.-H. A Comparison of Data Forwarding Schemes for Network Resiliency in Software Defined Networking. *Procedia Comput. Sci.* **2014**, *34*, 680–685. [CrossRef]
76. Tarnaras, G.; Haleplidis, E.; Denazis, S. SDN and ForCES based optimal network topology discovery. In Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft), London, UK, 13–17 April 2015; pp. 1–6. [CrossRef]
77. Rajaratnam, A.; Kadikar, R.; Prince, S.; Valarmathi, M. Software defined networks: Comparative analysis of topologies with ONOS. In Proceedings of the 2017 International Conference on Wireless Communications, Signal Processing and Networking, Chennai, India, 22–24 March 2017; pp. 1377–1381. [CrossRef]
78. Bah, M.T.; Del-Piccolo, V.; Bourguiba, M.; Haddadou, K. A centralized controller to improve fault tolerance in TRILL-based fabric networks. In Proceedings of the 2016 3rd Smart Cloud Networks & Systems (SCNS), Dubai, United Arab Emirates, 19–21 December 2016; pp. 1–6. [CrossRef]
79. Alharbi, T.; Portmann, M.; Pakzad, F. The (In)Security of Topology Discovery in Software Defined Networks. In Proceedings of the 2015 IEEE 40th Conference on Local Computer Networks, Clearwater Beach, FL, USA, 26–29 October 2015; pp. 502–505. [CrossRef]
80. Nguyen, T.-H.; Myungsik, Y. Analysis of link discovery service attacks in SDN controller. In Proceedings of the 2017 International Conference on Information Networking (ICOIN), Da Nang, Vietnam, 11–13 January 2017; pp. 259–261. [CrossRef]
81. Smyth, D.; McSweeney, S.; O'Shea, D.; Cionca, V. Detecting Link Fabrication Attacks in Software-Defined Networks. In Proceedings of the 2017 26th International Conference on Computer Communication and Networks (ICCCN), Vancouver, BC, Canada, 31 July–3 August 2017; pp. 1–8. [CrossRef]
82. Jiménez, Y.; Cervelló-Pastor, C.; García, A. Dynamic resource discovery protocol for software defined networks. *IEEE Commun. Lett.* **2015**, *19*, 743–746. [CrossRef]
83. Choi, J.S.; Kang, S.; Lee, Y. Design and evaluation of a PCEP-based topology discovery protocol for stateful PCE. *Opt. Switch. Netw.* **2017**, *26*, 39–47. [CrossRef]
84. Montero, R.; Agraz, F.; Pagès, A.; Perelló, J.; Spadaro, S. SDN-based parallel link discovery in optical transport networks. *Trans. Emerg. Telecommun. Technol.* **2019**, *30*, 1–13. [CrossRef]
85. Alvarez-Horcajo, J.; Rojas, E.; Martinez-Yelmo, I.; Savi, M.; Lopez-Pajares, D. HDDP: Hybrid Domain Discovery Protocol for Heterogeneous Devices in SDN. *IEEE Commun. Lett.* **2020**, *24*, 1655–1659. [CrossRef]
86. Zhao, R.; Liu, Y.; Dobre, O.A.; Wang, H.; Shen, X. An efficient topology discovery protocol with node id assignment based on layered model for underwater acoustic networks. *Sensors* **2020**, *20*, 6601. [CrossRef]
87. Tarnaras, G.; Athanasiou, F.; Denazis, S. Efficient topology discovery algorithm for software-defined networks. *IET Netw.* **2017**, *6*, 157–161. [CrossRef]
88. Katz, D.; Ward, D. RFC 5880, Bidirectional Forwarding Detection (June 2010).
89. Rifai, M.; Huin, N.; Caillouet, C.; Giroire, F.; Moulierac, J.; Lopez Pacheco, D.; Urvoy-Keller, G. Minnie: An SDN world with few compressed forwarding rules. *Comput. Netw.* **2017**, *121*, 185–207. [CrossRef]
90. Panda, A.; Samal, S.S.; Turuk, A.K.; Panda, A.; Venkatesh, V.C. Dynamic Hard Timeout based Flow Table Management in Openflow enabled SDN. In Proceedings of the 2019 International Conference on Vision Towards Emerging Trends in Communication and Networking, Vellore, India, 30–31 March 2019. [CrossRef]
91. Isyaku, B.; Kamat, M.B.; Abu Bakar, K.B.; Mohd Zahid, M.S.; Ghaleb, F.A. IHTA: Dynamic Idle-Hard Timeout Allocation Algorithm based OpenFlow Switch. In Proceedings of the 2020 IEEE 10th Symposium on Computer Applications & Industrial Electronics, Penang, Malaysia, 18–19 April 2020; pp. 170–175. [CrossRef]
92. Xu, X.; Hu, L.; Lin, H.; Fan, Z. An adaptive flow table adjustment algorithm for SDN. In Proceedings of the 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City, Zhangjiajie, China, 10–12 August 2019; pp. 1779–1784. [CrossRef]
93. Kotani, D.; Okabe, Y. A Packet-In message filtering mechanism for protection of control plane in OpenFlow switches. *IEICE Trans. Inf. Syst.* **2016**, *E99D*, 695–707. [CrossRef]
94. Favaro, A.; Ribeiro, E.P. Reducing SDN/openflow control plane overhead with blackhole mechanism. In Proceedings of the 2015 Global Information Infrastructure and Networking Symposium (GIIS), Tunis, Tunisia, 28–30 October 2020; pp. 1–4. [CrossRef]
95. Leng, B.; Huang, L.; Qiao, C.; Xu, H.; Wang, X. FTRS: A mechanism for reducing flow table entries in software defined networks. *Comput. Netw.* **2017**, *122*, 1–15. [CrossRef]
96. Li, Q.; Huang, N.; Wang, D.; Li, X.; Jiang, Y.; Song, Z. HQTimer: A Hybrid Q-Learning-Based Timeout Mechanism in Software-Defined Networks. *IEEE Trans. Netw. Serv. Manag.* **2019**, *16*, 156–166. [CrossRef]

97. Yang, H.; Riley, G.F. Machine learning based flow entry eviction for OpenFlow switches. In Proceedings of the 2018 27th International Conference on Computer Communication and Networks (ICCCN), Hangzhou, China, 30 July–2 August 2018. [CrossRef]
98. Alowa, A.; Fevens, T. A dynamic recovery module for in-band control channel failure in software defined networking. In Proceedings of the 2020 6th IEEE Conference on Network Softwarization, Ghent, Belgium, 29 June–3 July 2020; pp. 209–217. [CrossRef]
99. Fan, W.; Yang, F. Centralized Trust-Based In-Band Control for SDN Control Channel. *IEEE Access* **2020**, *8*, 4289–4300. [CrossRef]
100. Asadujjaman, A.S.M.; Rojas, E.; Alam, M.S.; Majumdar, S. Fast Control Channel Recovery for Resilient In-band OpenFlow Networks. In Proceedings of the 2018 4th IEEE Conference on Network Softwarization and Workshops, Montreal, QC, Canada, 25–29 June 2018; pp. 232–236. [CrossRef]
101. Osman, M.; Nunez-Martinez, J.; Mangues-Bafalluy, J. Hybrid SDN: Evaluation of the impact of an unreliable control channel. In Proceedings of the 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Berlin, Germany, 6–8 November 2017; Volume 2017, pp. 242–246. [CrossRef]
102. Hwang, R.-H.; Tang, Y.-C. Fast Failover Mechanism for SDN-Enabled Data Centers. In Proceedings of the 2016 International Computer Symposium (ICS), Chiayi, Taiwan, 15–17 December 2016; pp. 171–176. [CrossRef]
103. Ko, K.; Son, D.; Hyun, J.; Li, J.; Han, Y.; Hong, J.W.-K. Dynamic failover for SDN-based virtual networks. In Proceedings of the 2017 IEEE Conference on Network Softwarization (NetSoft), Bologna, Italy, 3–7 July 2017; pp. 1–5. [CrossRef]
104. Ibrar, M.; Wang, L.; Muntean, G.; Akbar, A.; Shah, N.; Malik, K.R. PrePass-Flow: A Machine Learning based technique to minimize ACL policy violation due to links failure in hybrid SDN. *Comput. Netw.* **2021**, *184*, 107706. [CrossRef]
105. Gao, P.; Xu, Y.; Chao, H.J. OVS-CAB: Efficient rule-caching for Open vSwitch hardware offloading. *Comput. Netw.* **2021**, *188*, 107844. [CrossRef]