Álvarez Horcajo, J., Rojas, E., Martínez Yelmo, I., Savi, M. & López Pajares, D. 2020, "HDDP: Hybrid Domain Discovery Protocol for heterogeneous devices in SDN", IEEE Communications Letters, vol. 24, no. 8, pp. 1655-1659.

Available at http://dx.doi.org/10.1109/LCOMM.2020.2991347

HDDP: Hybrid Domain Discovery Protocol for heterogeneous devices in SDN

Joaquin Alvarez-Horcajo, Elisa Rojas, Isaias Martinez-Yelmo, Marco Savi and Diego Lopez-Pajares

***Abstract*—Computer networks are adopting the new Software-Defined Networking (SDN) architecture, however not all devices can support it, mainly due to power and computational constraints. This paper proposes the Hybrid Domain Discovery Protocol (HDDP), a new discovery protocol that enhances the existing OpenFlow Discovery Protocol (OFDP). HDDP allows the discovery of hybrid network topologies composed of both SDN and non-SDN devices, which no other state-of-the-art protocol can achieve. HDDP has been implemented in a software switch and emulated in diverse networks, where it discovers hybrid topologies by using a number of messages similar to competitors, as they only discover SDN devices.**

***Index Terms*—Hybrid, SDN, OpenFlow, Topology Discovery, LLDP, OFDP**

I. Introduction

THE next generation of mobile networks (5G) predicts the proliferation of end devices. Managing these resources is often complex because of their heterogeneity and particularly due to scalability concerns, e.g., sensors usually have battery and memory constraints, and, as such, typical control approaches based on SDN are not always feasible. Indeed, topology discovery is one of the key points in SDN solutions [1] because of the possibilities offered by the centralized control plane. Most SDN platforms are able to discover the topology of a whole network as long as they support OpenFlow or other Southbound Interface (SBI) protocols, while non-SDN resources are only partially discovered as end devices (with no specific topology) or directly omitted. However, in some cases, such as in fog computing environments (responsible for managing Internet-of-Things (IoT) sensors), guaranteeing exact connectivity and resource discovery is vital [2]. A service capable of discovering non-SDN devices hybridized in SDN environments would bring many benefits, including improved traffic engineering or debugging of the anomalous behavior.

In this article, we present HDDP, which provides an enhanced topology discovery service capable of obtaining the entire topology in hybrid SDN deployments with low control message overhead. To this purpose, we first examine the related work in Section II. Secondly, we describe, implement, and evaluate HDDP in Sections III, IV, and V, respectively. Finally, we conclude the analysis in Section VI.

Joaquin Alvarez-Horcajo, Elisa Rojas, Isaias Martinez-Yelmo and Diego Lopez-Pajares are with Departamento de Automatica, University ofAlcala, 28805, Alcala de Henares. Marco Savi is with University of Milano-Bicocca, Milano, Italy. The study was done while he was with Fondazione Bruno Kessler, Trento, Italy. Corresponding author's e-mail: elisa.rojas@uah.es

This work was funded by grants from Comunidad de Madrid through Project TAPIR-CM (S2018/TCS-4496) and from University of Alcalá through "Programa de Formación del Profesorado Universitario (FPU)" and the project CCG2018_EXP-076.

II. Related work

Nowadays, there is no standalone topology discovery solution that can obtain the full topology of a hybrid SDN domain with SDN and non-SDN devices. The most commonly deployed service for topology discovery in SDN is OFDP [3], based on Link Layer Discovery Protocol (LLDP). However, OFDP requires that all devices support OpenFlow [3]. Diverse protocols including OFDPv2 [4], Tree Exploration Discovery Protocol (TEDP) [5] or enhanced Topology Discovery Protocol (eTDP) [6] outperform OFDP in terms of scalability and efficiency, however, none of them is capable of discovering non-SDN devices. The first work to tackle the discovery of resources in hybrid networks was [7], which combined LLDP with Broadcast Domain Discovery Protocol (BDDP) to unveil legacy switches located between SDN switches. However, they lacked mechanisms to avoid broadcast storms without the use of Spanning Tree Protocol (STP). Such lack prevented a full discovery of the underlying hybrid topology since some links were removed to avoid loops. Hence, this proposal could not discover complex combinations of SDN, non-SDN devices, and even end nodes. Also, aiming at hybrid SDN discovery, the work in [8] presented a programmable SDN device, based on Forwarding and Control Element Separation (ForCES) [9], capable of detecting non-SDN neighbor nodes based on indirectly snooping traffic from traditional protocols (such as LLDP). The main advantage is that it does not require modifications of legacy devices. Nevertheless, it does not guarantee the full discovery of non-SDN topologies because either the indirect discovery might become too complex in large and/or heterogeneous networks, or could even prove impossible in some topologies (e.g., with loops of devices).

Software-Defined Wireless Sensor Networks (SDWSNs) are particularly challenging scenarios to apply SDN because OFDP has scalability issues due to their size [10] and their fast topology changes because of node mobility [11]. Thus, IoT Hub [12] introduced an edge manager for resource discovery; however, it is unable to discover topologies among the IoT devices. Authors of [13] proposed a strategy for the detection of heterogeneous network devices that relies on a direct connection between the devices and the SDN controller, which is not always feasible. Our proposal, HDDP, is especially suitable for SDWSNs as the non-SDN devices are mostly Linux-based systems on which it can be easily deployed.

A. Contributions of HDDP to the state of the art and SDWSNs

The main contribution of HDDP is that it can fully discover the underlying topology in hybrid environments (i.e., both

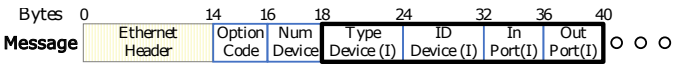


Fig. 1. HDDP Control Message

SDN and non-SDN devices, as well as the existing wireless connectivity among them), which, to our knowledge, no other protocol in the current state of the art can achieve.In the specific case of SDWSNs, HDDP makes it possible to know whether a sensor acts as a forwarding relay for other sensors, thus allowing the discovery of complex sub-topologies among non-SDN sensors. This allows to disclose (and later mitigate) any possible anomalous behavior in those portions of the network.

Additionally, given its exploration nature, HDDP is able to discover links between non-SDN devices that may not be currently used in the communication. This is especially useful to recognize if multiple gateways are in the coverage area of a sensor to later deduce how many sensors could be paired to each gateway. Such information can be very helpful for traffic engineering and load balancing purposes (e.g., to relieve a gateway from processing traffic if overloaded).

Finally, HDDP can be used to monitor the mobility of sensors within the SDWSN. This information is important for different network management and traffic engineering tasks, especially when the SDWSN has edge/fog computing capabilities [14] and can perform some computation on the data flows [15]. When mobility occurs, the computation of SDWSN data flows could be rearranged on the edge/fog infrastructure to be optimized, according to the current location of the sensors.

III. Definition of HDDP

The main feature of HDDP is the ability to incorporate non-SDN devices and their bidirectional links into a full hybrid SDN network discovery. As Non-SDN devices cannot have direct links to SDN controllers, they rely on the SDN devices to report their information (non-SDN devices may not have a direct connection to an SDN device, but an indirect one through other non-SDN devices). HDDP is a distributed protocol that relies on the exchange of *HDDP Request* and *Reply* control messages to unveil the network topology. Figure 1 shows the structure of an HDDP control message (it has variable length depending on the value of the *Num Device* field). This structure is based on the minimum length of Ethernet networks as it is the infrastructure layer chosen for the implementation of HDDP, although it could be reduced in other infrastructures (e.g., wireless scenarios). Figures 2 and 3 show how the controller orders its attached SDN devices to broadcast an *HDDP Request* message to explore the network, which is followed by the corresponding *HDDP Reply* messages from all devices. The devices convey the information required by the controller to obtain a full view of the underlying topology by combining different data subsets extracted from these messages. The key point is how HDDP triggers these *HDDP Reply* messages, which depends on the kind of nodes (wired or wireless).

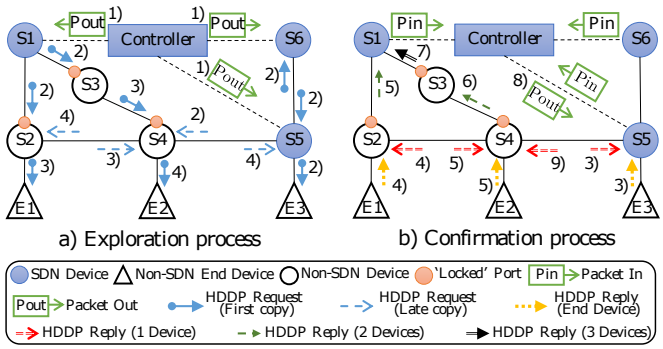


Fig. 2. HDDP behavior example in a wired environment

A. HDDP in wired scenarios

This section describes the behavior of HDDP in wired scenarios, which are the ones considered in this paper for implementation and evaluation. As an example, we consider the random wired topology shown in Fig. 2 to explain how HDDP discovers the proposed hybrid topology. This hybrid topology has three SDN switches[1] (where the controller has installed a rule to indicate that all HDDP control messages must be sent to the controller), three non-SDN switches (e.g., gateways), and three non-SDN end devices (e.g., hosts). As Fig. 2a illustrates, the exploration process is triggered by the SDN controller, which broadcasts a *PACKET_OUT* message to all SDN devices within the network domain (see step 1) by encapsulating an *HDDP Request* message. The SDN controller builds the *HDDP Request* message by setting the values of *Option Code* and *Num Device* fields to 1, *ID Device* field to the Datapath ID (DpId) of each SDN node, and finally, the *In-Port* and *Out-Port* fields to 0. When these packets are received by any SDN node, it decapsulates and broadcasts the *HDDP Request* message through all its ports (see steps marked as 2).

On the one hand, the non-SDN nodes receiving the *HDDP Request* message lock the first entry port to avoid loops, according to the All-Path locking mechanism [16]. Then, they increment the *Num Device* field of the *HDDP Request* message by one (see Fig. 1) and forward it again via all their ports except the ingress one (steps 3 and 4). As illustrated by Fig. 2b, late copies of the *HDDP Request* message, arriving at other ports different from the initially locked one, are used to trigger the sending of *HDDP Reply* messages backward through the late *HDDP Request* copy incoming port (steps 4 and 5 in dashed arrows). An *HDDP Reply* message is built by updating a copy of the *HDDP Request* message, which is transformed into an *HDDP Reply* message by changing the *Option Code* value to 2 and the *Num Device* data to 1. Later on, the nodes insert a new tuple composed of *(Type Device, ID Device, In-Port, and Out-Port)* (highlighted in bold in Fig. 1). The *Type Device* field encodes whether the node is a switch, gateway, or host, including its functionality (e.g., gateway with computing capabilities). The *ID Device* field is the DpId for the SDN devices and the MAC address converted to an unsigned long integer value for the non-SDN devices. The *In-Port* field is

[1]Note the communication with the controller might be out-of-band or in-band, as HDDP does not depend on it.

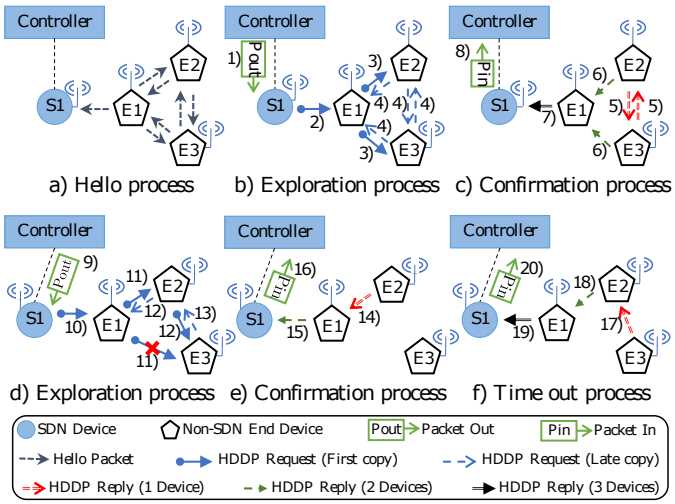
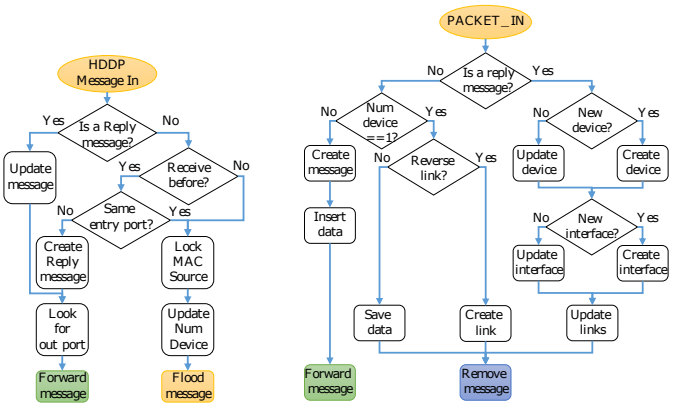


Fig. 3. HDDP behavior example in a wireless environment



(a) Non-SDN Device (b) SDN Controller

Fig. 4. HDDP components

the input port of the *HDDP Request* message and the *Out-Port* field is the output port of the *HDDP Reply* message itself, which are required in the controller to build the hybrid topology. The non-SDN nodes receiving the *HDDP Reply* message increase the *Num Device* field by one and insert their own tuple. The *Type Device* and *ID device* fields are set as described previously. However, the *In-Port* field is set to the incoming port of the *HDDP Reply* message and the *Out-Port* field is set to the input port of the first received *HDDP Request* message (steps 5, 6, and 7). These *HDDP Reply* messages are sent through the port specified in the *Out-Port* field. When the *HDDP Reply* messages turn back to any SDN node, they forward such packets via *PACKET_IN* messages to the SDN controller, which collects all the gathered information across the non-SDN devices. The non-SDN end-devices connected as leaf nodes send an *HDDP Reply* message when they receive an *HDDP Request* message via their only port (steps 3-5).

On the other hand, the SDN nodes receiving the *HDDP Request* message forward it to the controller via a *PACKET_IN* message, similarly to OFDP. The controller checks if the *Num Device* field is one: if so, it means that two SDN devices are neighbors. However, if it is higher than one, it means that the number of non-SDN devices between two SDN devices is *Num Device* minus one. Thus, the controller replies with an *HDDP Reply* message in a *PACKET_OUT* message using the same port as it happens for non-SDN end devices. Such *HDDP Reply* message collects the information from non-SDN devices, and guarantees all links between SDN and non-SDN nodes are discovered (steps 8-9).

Finally, the controller gets the full hybrid topology from the data embedded in the *HDDP Reply* messages.

B. HDDP in wireless scenarios

In the case of wireless (or mixed wired-wireless) networks, such as SDWSNs, there are two main differences to consider: (1) the locking mechanism cannot be applied to a port since sensors usually have a single wireless interface, and (2) the number of neighbors is unknown, especially if node mobility is admitted. For the first difference, we just need to consider that the locking mechanism basically locks an identifier of the neighbor that first sent an *HDDP Request* message. In wired networks, the identifier is the input port (the port is unique per neighbor), but in wireless networks the identifier for the locking could be the MAC address of the previous hop/neighbor since this information is transmitted in the frame and can be easily collected. Regarding the second difference, to distinguish when a node has only one neighbor (to send an *HDDP Reply*) or more (to send an *HDDP Request*), there are two possible mechanisms to apply: (1) wireless nodes periodically send *Hello* messages to identify and acknowledge their neighbors; (2) wireless nodes broadcast the *HDDP Request* message by default and, afterwards, if they do not receive the response of an *HDDP Reply* message, they generate their own *HDDP Reply* message after a timeout. Note that if the timer expires before the expected message is received, the functionality of HDDP is not affected since additionally generated *HDDP Reply* messages just provide more topological information.

Figure 3 illustrates an example with one SDN node ($S1$) and three non-SDN sensors ($E1$, $E2$, and $E3$) to clarify how HDDP operates in wireless scenarios. As depicted in Fig. 3a, the non-SDN wireless nodes generate periodic *Hello* messages that are received by all the nodes in their range. Afterwards, Fig. 3b and Fig. 3c represent the usual exchange of *HDDP Request* and *HDDP Reply* messages, respectively, as explained in the previous section (steps 1-8). In the hypothetical situation in which, for example, node $E3$ moves, this is noticed by the periodic exchange of *Hello* messages. If this is not detected because the *Hello message* does not arrive in time, Fig. 3d, Fig. 3e and Fig. 3f show what happens. Firstly, $E3$ only receives one *HDDP Request* message (from $E2$) as shown in step 12. In the meantime, $E3$ waits for more messages while $E2$ sends the *HDDP Reply* message to $E1$ due to a timeout, as depicted in step 14. Finally, in step 17 the timer of $E3$ expires as well. Hence, $E3$ also sends an *HDDP Reply* message to $E2$, which forwards it again to $E1$ to complete the missing topological information (steps 18-19).

IV. HDDP IMPLEMENTATION

The implementation of HDDP requires two different software components. The component of HDDP in non-SDN

network devices (Fig. 4a) and the component for processing the HDDP control messages in the SDN controller (Fig. 4b).

A. HDDP support in non-SDN devices

Figure 4a summarises the proof-of-concept implementation based on the *ofsofswitch* software switch [17]. The first step is to determine the type of HDDP control message by checking the value of the option code (*OpCode* field), which confirms whether it is an *HDDP Request* or an *HDDP Reply* message. If the frame is an *HDDP Request* message, the device checks if the source MAC address has been received before. If a match exists and the input port is the same as the first input port, the lock time of the source MAC address entry is updated and the value of the *Num Device* field is increased by one unit; finally, the packet can be broadcasted. Otherwise, an *HDDP Reply* message is sent following the structure shown in Fig. 1 via the incoming port of the *HDDP Request* message. In case that the source MAC address is not locked, the node must lock the in-port, increase the *Num Device* field by one and broadcast the frame. If the frame is an *HDDP Reply* message, the field *Num Device* is increased by one unit and the new structure shown in Fig. 1 is inserted at the end of the frame. Finally, the node uses the locked port as output port to forward the frame.

B. HDDP support in ONOS platform

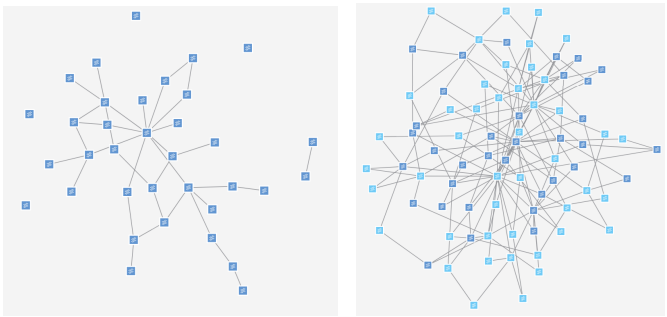
Figure 4b summarises the proof-of-concept implementation developed in Open Network Operating System (ONOS) to handle HDDP messages and to build the topology from them. Periodically, ONOS must send *HDDP Request* messages through *PACKET_OUT* messages and later may receive them as *PACKET_IN* messages (depending on the underlying topology). If an *HDDP Request* message is received, the value of the *Num Device* is obtained. If it is equal to one, the information is saved until the *HDDP Request* message in the opposite direction is received to create the link between both devices. Otherwise, if *Num Device* is more than one, it is necessary to send an *HDDP Reply* message with the carrying tuple *(Device Type, ID Device, In-Port, Out-Port)* through a *PACKET_OUT* message, as stated in the previous section. Finally, if an *HDDP Reply* message is received, the information about new devices and links is included in the topological database, and the previously existing devices and links are updated.

V. Evaluation and Discussion

The evaluation of HDDP was performed with an infrastructure consisting of several hosts with Intel(R) Core(TM) i7 CPUs and 24 GB of RAM, running ONOS as SDN controller, and Mininet to emulate the GÉANT pan-European network [18] and different random Barabási-Albert networks [19]. Each experiment was repeated 10 times, and the mean and confidence interval for the results in each experiment were computed. The evaluation parameter that we selected was the number of control messages, which can be easily translated into bandwidth usage in *SDN-only* networks as the HDDP message size is fixed (only conveys data for one device).

TABLE I
Results comparison of LLDP, TEDP and HDDP

Packet#		Control Plane						Data Plane	
		Packet_In		Packet_Out		Flow_Mod			
		Theory	Real	Theory	Real	Theory	Real	Theory	Real
LLDP	$\bar{x}$	144	145	144	160.06	44	44	144	160.06
	σ	-	14.24	-	19.78	-	0	-	19.78
TEDP	$\bar{x}$	144	170.63	1	1	88	227.44	144	170.63
	σ	-	2.31	-	0	-	1.71	-	2.31
HDDP	$\bar{x}$	144	144.72	44	44	44	44	144	144.97
	σ	-	1.31	-	0	-	0	-	1.22



(a) OFDP topology (b) HDDP topology

Fig. 5. The same topology discovered by OFDP and HDDP

A. SDN topology discovery

To compare HDDP with the state of the art, we first deployed the GÉANT pan-European network topology composed of 44 SDN nodes and a total of 144 ports (72 links). We calculated the number of packets needed by LLDP, TEDP, and HDDP, to discover the topology, and compared them in an SDN-only scenario. The results are summarised in the *Theory* columns of Table I. We can see how the number of *PACKET_IN* messages used by LLDP, TEDP and HDDP are equal, and LLDP and HDDP produce the same amount of *FLOW_MOD* messages. Furthermore,HDDP reduces the number of *PACKET_OUT* messages by 72.5% regarding LLDP, as it only needs a *PACKET_OUT* message per SDN node while LLDP needs a *PACKET_OUT* message per each port in all SDN nodes in the domain. However, the number of *PACKET_OUT* messages is smaller for TEDP because it just sends one packet to start the discovery exploration process, while HDDP sends a *PACKET_OUT* message per node in the topology. Moreover, Table I also shows how the number of packets exchanged in the data plane is the same in all approaches (one per port). To confirm the analytical results, we measured the protocol behavior of both protocols using the GÉANT pan-European topology in Mininet. The results in the *Real* columns of Table I demonstrate the accuracy of the previous analytical results.

B. Hybrid topology discovery

The main advantage of HDDP is that it can discover the full topology in hybrid networks where SDN and non-SDN nodes coexist. The only requirement is the support of HDDP in non-SDN devices, as well as in the controller. For this study, we used random Barabási-Albert topologies to evaluate HDDP on an extensive basis. These topologies were selected with a different number of nodes, degrees and percentage of SDN devices to check the behaviour of HDDP. As an example,

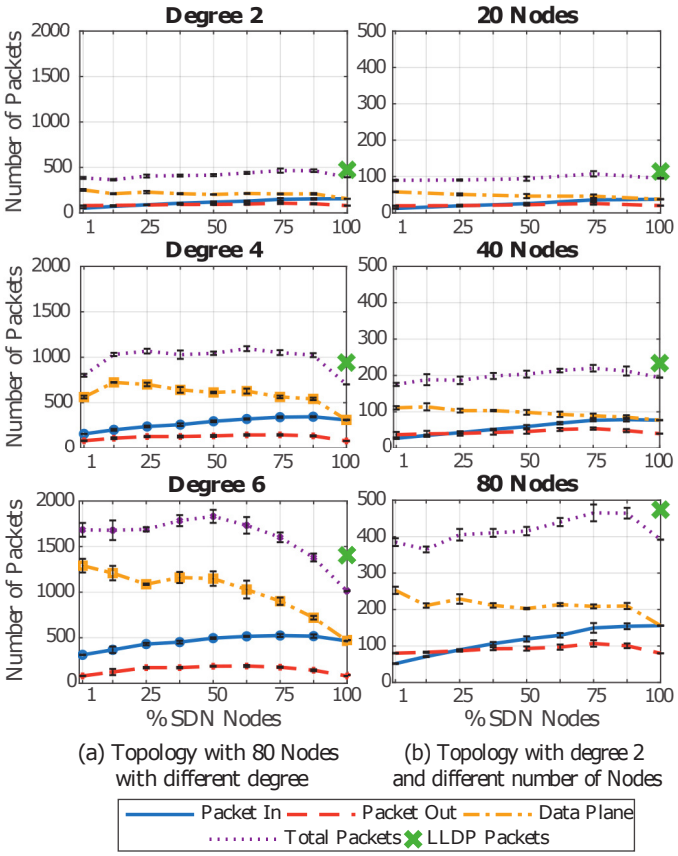


Fig. 6. Number of packets in Barabási-Albert topologies

Fig. 5a shows how OFDP can discover only SDN nodes (dark blue elements in the figure) and a subset of the links, while Fig. 5b illustrates how HDDP can obtain all nodes (non-SDN nodes are in light blue) and links.

Specifically, the performance of HDDP is evaluated by measuring the number of messages needed to discover different Barabási-Albert topologies with different nodal degrees in an 80-node topology (Fig. 6a), and in diverse topologies with an average node degree of two and an increasing number of nodes (Fig. 6b). In both cases, the percentage of SDN devices ranges from 0 to 100% in the topology. The results show how the required number of packet increases, as the degree rises (Fig. 6a). This increase is mainly caused by the HDDP data plane traffic, which is required to discover the non-SDN devices. Moreover, the behaviour of HDDP is closer to LLDP as the number of SDN nodes increases, as expected. Figure 6b also shows how the number of packets increases as the number of nodes rises. Once again, as the percentage of SDN devices increases, HDDP performance is comparable to the LLDP one. Additionally, HDDP always obtains the whole hybrid topology (all nodes and all links).

VI. Conclusion

This paper presented HDDP, an innovative protocol able to accomplish a full topology discovery in hybrid SDN domains, where SDN and non-SDN devices coexist. The protocol is based on an exploration mechanism triggered by the control plane, which reaches all devices via a controlled flooding mechanism to collect all the necessary information from hybrid networks. Our proposal outperforms OFDP since it is capable of discovering a full hybrid topology from a network domain composed of SDN and non-SDN devices with fewer exchanged packets. In our future work, we would like to optimize the use of control messages, considering how their periodicity affects scenarios with high mobility, analyze the implication of unidirectional links in wireless scenarios, and evaluate the diverse security implications of this approach.

References

[1] G. Tarnaras *et al.*, "SDN and ForCES based Optimal Network Topology Discovery," in *IEEE Conference on Network Softwarization*, 2015.
[2] R. Mahmud *et al.*, "Fog Computing: A Taxonomy, Survey and Future Directions," in *Springer Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives*, pp. 103–130, 2018.
[3] D. Hasan and M. Othman, "Efficient Topology Discovery in Software Defined Networks: Revisited," *Elsevier Procedia Computer Science*, vol. 116, pp. 539–547, 2017.
[4] F. Pakzad *et al.*, "Efficient Topology Discovery in OpenFlow-based Software Defined Networks," *Elsevier Computer Communications*, vol. 77, no. C, pp. 52–61, 2016.
[5] E. Rojas *et al.*, "TEDP: An Enhanced Topology Discovery Service for Software-Defined Networking," *IEEE Communications Letters*, vol. 22, no. 8, pp. 1540–1543, 2018.
[6] L. Ochoa-Aday *et al.*, "eTDP: Enhanced Topology Discovery Protocol for Software-Defined Networks," *IEEE Access*, vol. 7, pp. 23 471–23 487, 2019.
[7] L. Ochoa-Aday *et al.*, "Current Trends of Topology Discovery in OpenFlow-based Software Defined Networks," Tech. Rep., 2015. [Online]. Available: https://upcommons.upc.edu/handle/2117/77672
[8] G. Tarnaras *et al.*, "Efficient Topology Discovery Algorithm for Software-Defined Networks," *IET Networks*, vol. 6, pp. 157–161, 2017.
[9] J. Halpern and J. H. Salim, "Forwarding and Control Element Separation (ForCES) Forwarding Element Model," IETF, RFC 5812, 2010. [Online]. Available: http://tools.ietf.org/rfc/rfc5812.txt
[10] J. Kipongo *et al.*, "Topology Discovery Protocol for Software Defined Wireless Sensor Network: Solutions and Open Issues," in *IEEE International Symposium on Industrial Electronics*, 2018.
[11] S. Babu *et al.*, "A Novel Framework for Resource Discovery and Self-Configuration in Software Defined Wireless Mesh Networks," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 132–146, 2019.
[12] S. Cirani *et al.*, "The IoT Hub: A Fog Node for Seamless Management of Heterogeneous Connected Smart Objects," in *IEEE International Conference on Sensing, Communication, and Networking - Workshops*, 2015.
[13] S. K. Panda *et al.*, "Topology Detection as a Base for Efficient Management of Heterogeneous Industrial Network Systems Using Software-Defined Networking," in *IEEE International Workshop on Factory Communication Systems*, 2019.
[14] A. Yousefpour *et al.*, "All One Needs to Know About Fog Computing and Related Edge Computing Paradigms: A Complete Survey," *Elsevier Journal of Systems Architecture*, vol. 98, pp. 289–330, 2019.
[15] T. Rahman *et al.*, "Efficient Edge Nodes Reconfiguration and Selection for the Internet of Things," *IEEE Sensors Journal*, vol. 19, no. 12, pp. 4672–4679, 2019.
[16] E. Rojas *et al.*, "All-Path Bridging: Path Exploration Protocols for Data Center and Campus Networks," *Elsevier Computer Networks*, vol. 79, pp. 120 – 132, 2015.
[17] E. Leão Fernandes and C. Esteve Rothenberg, "OpenFlow 1.3 Software Switch," in *Salão de Ferramentas XXXII Simpósio Brasileiro de Redes de Computadores*, 2014.
[18] "GÉANT Topology Map August 2017," Tech. Rep., 2017. [Online]. Available: https://www.geant.org/Resources/PublishingImages/GEANT_topology_map_august2017.pdf
[19] A. L. Barabási and R. Albert, "Emergence of Scaling in Random Networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.