Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

eHDDP: Enhanced Hybrid Domain Discovery Protocol for network topologies with both wired/wireless and SDN/non-SDN devices[☆]

Isaias Martinez-Yelmo[*], Joaquin Alvarez-Horcajo, Juan Antonio Carral, Diego Lopez-Pajares

University of Alcalá, Pza. San Diego, s/n, 28801, Alcalá de Henares (Madrid), Spain

ARTICLE INFO

Keywords:
Hybrid
SDN
WSN
IoT
Discovery
Performance
Protocol
Unidirectional
Bidirectional

ABSTRACT

Handling efficiently both wired and/or wireless devices in SDN networks is still an open issue. eHDDP comes as an enhanced version of the Hybrid Domain Discovery Protocol (HDDP) that allows the SDN control plane to discover and manage hybrid topologies composed by both SDN and non-SDN devices with wired and/or wireless interfaces, thus opening a path for the integration of IoT and SDN networks. Moreover, the proposal is also able to detect both unidirectional and bidirectional links between wireless devices. eHDDP has been thoroughly evaluated in different scenarios and exhibits good scalability properties since the number of required messages is proportional to the number of existing links in the network topology. Moreover, the obtained discovery and processing times give the opportunity to support scenarios with low mobility devices since the discovery times are in the range of hundreds of milliseconds.

1. Introduction

Data communication networks are a hot topic since the birth of Internet which have been on continuous evolution until now. Nowadays, this evolution is even stronger, and embraces many different aspects such as the definition of new networking paradigms like the SDN architecture or the Internet-of-Things (IoT). Although SDN networks have many advantages and functionalities [1], the cost of the necessary investments to replace previous deployments is preventing a faster penetration in current networks. On the other hand, the IoT is growing very fast due to its use in new services, appliances and personal devices. However, IoT devices have important hardware and software constraints which limit its integration with SDN networks despite the promised benefits regarding availability of new services and ease of management of the whole network.

Hybrid Domain Discovery Protocol (HDDP) [2] is the first work capable of fully discovering hybrid SDN network topologies with both wired SDN devices (those capable of direct interaction with the SDN control plane via OpenFlow, P4Runtime, etc.) and non-SDN devices (those unable to interact with the SDN control plane in anyway) by using a lightweight agent that implements HDDP and sends indirectly the topological information to the SDN control plane. HDDP follows the network exploration paradigm based on controlled flooding of messages to discover non-SDN devices and the same approach as OpenFlow Discovery Protocol version 2 (OFDPv2) [3] to discover neighbour SDN devices. Moreover, the work in HDDP envisions different approaches to allow its use in wireless scenarios which is the main objective of this paper.

This paper presents eHDDP, an enhanced version of HDDP, which allows its use not only in wired interfaces but also in wireless interfaces, including low powered wireless technologies, in an efficient way. Due to the proposed enhancements, eHDDP is the first proposal offering a general solution to the problem of discovering fully hybrid SDN network topologies comprised of both SDN and non-SDN devices with wired and/or wireless interfaces to the best of our knowledge. Some interesting use cases of the discovery of wireless hybrid SDN networks are the integration of Wireless Sensor Networks (WSNs) or the IoT networks by the SDN control plane to manage and supervise them in a more accurate way. The main contributions of this paper are:

- The upgrade of the HDDP header in the control messages to support different types of wireless interfaces. The header fields have been revised to minimise its size to support technologies with small Maximum Transfer Unit (MTU) size.
- The definition of an additional protocol logic to deal with wireless interfaces considering the number of neighbours is always unknown and may change with time (the protocol logic for wired

☆ This work was funded by grants from Comunidad de Madrid through Project TAPIR-CM, Spain (S2018/TCS-4496) and Project IRIS-CM, Spain (CM/JIN/2019-039), from Junta de Comunidades de Castilla la Mancha through Project IRIS-JCCM, Spain (SBPLY/19/180501/000324) and from University of Alcalá, Spain through "Programa de Formación del Profesorado Universitario (FPU)".

* Corresponding author.
E-mail address: isaias.martinezy@uah.es (I. Martinez-Yelmo).

https://doi.org/10.1016/j.comnet.2021.107983
Received 15 July 2020; Received in revised form 24 February 2021; Accepted 2 March 2021
Available online 8 March 2021

interfaces is the same as the one proposed for wired interfaces in HDDP).
- The detection of uni and bidirectional links in wireless scenarios.
- A reference implementation of eHDDP comprised of a network agent in non-SDN devices and an application in the SDN control plane by using Open Network Operating System (ONOS) and the OpenFlow protocol.
- The enhancement of previously existing network emulation tools to properly deploy and perform experiments with wireless Ad-Hoc networks with multiple devices.
- A thorough evaluation of eHDDP under different types of network topologies: wired, wireless and both wired and wireless topologies.

The rest of the paper is structured as follows. Section 2, Related Work, presents previous works related to topology discovery solutions in SDN and legacy networks as well as in wireless networks. Section 3 is devoted to providing a detailed explanation of the new proposal and is followed by Section 4, which presents the protocol logic and details on the prototype implementation developed for evaluation purposes. Section 5 describes the testbed that has been used to conduct the different experiments as well as all the enhancements that has been developed to adapt existing tools, such as Mininet, to setup the evaluation platform for wired and wireless hybrid SDN networks. Finally, Section 6 presents the exhaustive evaluation conducted on eHDDP and Section 7 summarises the conclusions derived from this work and envisions the future work.

2. Related work

This section is structured as follows. First, we provide an overview of discovery protocols for wired networks, focusing on those used in the SDN context. Second, we review the literature that tackled the discovery in wired hybrid SDN networks. Third, we summarise the proposals to support SDN capabilities in wireless devices, which include network discovery mechanisms in Software-Defined Wireless Sensor Networks (SDWSNs) to provide the necessary information to the SDN control plane in order to perform its duties in wireless scenarios. Finally, we conclude explaining the aims of eHDDP.

The design of network topology discovery protocols is a classical topic in the field of wired switched networks and the Internet. Probably, the most popular discovery protocols for legacy distributed networks are the standardised Link Layer Discovery Protocol (LLDP) [4], or Cisco Discovery Protocol (CDP). Nevertheless, discovery protocols are specially relevant in SDN networks because the control plane usually requires a full knowledge of the underlying network topology to accomplish its entrusted tasks. Among the different existing proposals, it is worth to mention OpenFlow Discovery Protocol (OFDP) [5], which is an adaption of LLDP for topology discovery in SDN networks but it presents scalability issues [6]. OFDP is enhanced by secure OpenFlow Topology Discovery Protocol (sOFTDP) [6] but it requires modifications in the OpenFlow protocol to work properly. Other works that enhance OFDP are OFDPv2 [3], which reduces the control plane load by slightly modifying LLDP control frames and preinstalling certain rules in SDN devices, whereas Software Defined - Topology Discovery Protocol (SD-TDP) [7] enhances OFDP by ordering SDN devices in a hierarchical structure. Additionally, there is an interesting survey on topology discovery solutions for SDN networks [8] but it does not include the most recent proposals such as Tree Exploration Discovery Protocol (TEDP) [9] or enhanced Topology Discovery Protocol (eTDP) [10]. TEDP is a topology discovery protocol that also provides latency-based paths among the SDN devices in a network by consuming a similar amount of control packets as OFDPv2. To conclude, eTDP outperforms OFDPv2 in packet consumption by taking advantage of the shortest control path towards each switch.

None of the previous proposals tackles the problem of hybrid SDN networks in which SDN and other devices coexist. Nevertheless, this can happen due to several factors, e.g., partial migration to SDN technology because of high costs and/or lack of resources (CPU, power, memory) in certain devices. Therefore, if the non-SDN devices and their connectivity were known in hybrid SDN network topologies, the SDN control plane could manage of all existing devices including those without SDN support by using other mechanisms, e.g., via Simple Network Management Protocol (SNMP). The first work in addressing this problem is [11], which combines LLDP and Broadcast Domain Discovery Protocol (BDDP) to discover legacy devices (devices before the definition of the SDN architecture with distributed functionalities) in a hybrid SDN network. This solution is not valid for topologies containing loops of non-SDN devices. It is also worth mentioning [12] since it makes use of Forwarding and Control Element Separation (ForCES) [13] to detect non-SDN neighbour devices. Non-SDN devices do not need to be modified if they already support ForCES, but the proposal can find problems in large or heterogeneous networks since its passive mechanism based on LLDP and Address Resolution Protocol (ARP) cannot deal with topologies containing loops of non-SDN devices.

Moreover, there is a relevant research line focusing on integrating SDN networks with IoT and Wireless Sensor Network (WSN) networks named as Software-Defined Wireless Sensor Network (SDWSN)s. Although there are proposals such as FlowSensor [14] or Sensor OpenFlow [15], they only simply integrate OpenFlow in wireless devices but do not solve the existing scalability [16] or mobility [17] issues. Some proposals in the same line are TinySDN [18] and SD-Wise [19], but they lack support for unidirectional wireless links, and other proposals have important design constraints [20,21]. One of the first works supporting the discovery of unidirectional links in SDWSNs is the work in [22]. However, it is limited to only wireless links, and its design may limit its applicability to track device mobility.

eHDDP enhances HDDP by supporting wireless interfaces and networks (the operation in wired networks is exactly the same as HDDP). Furthermore, it solves most of the aforementioned open issues. It focuses on the support of fully hybrid SDN heterogeneous network topologies featuring both wired and wireless links including even topologies with loops since it does not infer topological information from legacy protocols. It uses a proactive method based on network exploration instead. Although the obtained results allow the tracking of low mobility devices, a precise management of device mobility remains an open issue.

3. eHDDP: extended hybrid domain discovery protocol

The main objective of this work is to enhance HDDP to seamlessly deal with wireless interfaces and/or devices, as well as wired ones, in order to provide the SDN controller with a complete view of the underlying topology. As with HDDP, it requires the installation of a lightweight eHDDP agent on non-SDN devices since they cannot establish direct connection with the SDN control plane. eHDDP only requires an SDN control protocol that allows user data forwarding between the SDN control plane and the SDN devices of the data plane. The OpenFlow terminology is used along the paper since it is the most widely used protocol and the one used in the experiments. It is assumed that an out-of-band control channel between the SDN devices and the SDN control plane exists, which allows the discovery of SDN devices via southbound interfaces, e.g., the standard exchange of *OpenFlow Hello* messages.

eHDDP works in two phases: an exploration phase, started at the network controller, which relies on the broadcast of a *Request* control message to explore the whole network, and, a confirmation phase, based on *Reply* messages, designed to convey the information discovered, back to the control plane, from network devices. An additional control message (*ACK*) is required to confirm link bidirectionality in wireless mode. By combining the different topology subsets extracted from *Reply* messages, the control plane is able to construct a full view of

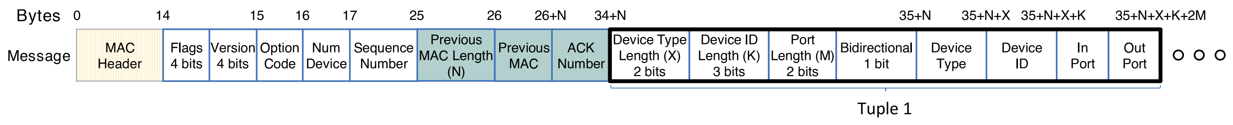


Fig. 1. eHDDP Control Message.

the underlying topology. The logic governing the confirmation phase of the protocol (*Reply* messages) depends on the type of interface (wired or wireless) and is thoroughly explained in the next subsections after a brief description of the control message structure.

3.1. eHDDP control message structure

eHDDP control messages follow the message structure shown in Fig. 1. It is composed of a *Common Header* followed by as many *Device Group* tuples (depicted in bold in the figure) as devices previously visited in *Reply* messages.

The eHDDP message structure has been redesigned to support fields of variable length for Media Access Control (MAC) addresses, Identifiers (IDs) or Types in order to decrease the size of the embedded topological information in low powered wireless technologies with small MTU and small MAC/ID/Type fields. Moreover, it considers some optional flags to only include those fields that are necessary in each case.

The different fields of the Common Header depicted in the figure are used as follows:

- *Flags*: this is a 4-bit field. The first 2 bits are Reserved bits, the third one shows whether exists the Acknowledgement field (Acknowledgement On), and the fourth one indicates if the control message has the Previous MAC field activated (Previous MAC On).
- *Version*: this is a 4-bit field. It indicates the protocol version being used.
- *Option Code*: this is an 8-bit field. It indicates the eHDDP message type. It can be a *Request* (an exploration packet), a *Reply* (a response from devices towards the controller), or an *ACK* (a special confirmation message designed to check link bidirectionality between neighbours in wireless interfaces).
- *Num Device*: this is an 8-bit field. It indicates the number of devices that the message has visited so far.
- *Sequence Number*: this is a 64-bit field. It is a unique random number identifying every control message of a discovery process iteration (Requests, Replies and ACKs). It is set in the original *Request* message by the controller when it starts the discovery process.
- *Previous MAC Length, N (Optional field)*: this is an 8-bit field. It indicates the Previous MAC field length in bytes.
- *Previous MAC (Optional field)*: this is a variable length field. It is the MAC address of the last device that sent the message.
- *Acknowledgement Number (Optional field)*: this is a 64-bit field. It is a random number linking a *Reply/ACK* pair of messages. It is set by the device sending a Reply message and copied in the corresponding *ACK* response.

The different fields of a *Device Group* tuple are as follows:

- *Device Type Length, X*: this is a 2-bit field. It indicates the Device Type field length in multiples of 2 bytes.
- *Device ID Length, K*: this is a 3-bit field. It indicates the Device ID field length in bytes.
- *Port Length, M*: this is a 2-bit field. It indicates the length of the In Port and Out Port fields in bytes.
- *Bidirectional*: this is a 1-bit field. It indicates whether the link identified by In Port is bidirectional.
- *Device Type*: this is a variable length field. It indicates whether the device is a switch, gateway, sensor or host, and it also includes some information about its functionality (e.g., gateway with computing capabilities).
- *Device ID*: this is a variable length field. It is the DPID for the SDN devices, the MAC address converted to an unsigned long integer value for the non-SDN devices with MAC address, and the device ID in the remaining non-SDN devices.
- *In Port*: this is a variable length field. It indicates the incoming interface of the message.
- *Out Port*: this is a variable length field. It indicates the outgoing interfaces of the message.

The eHDDP control message uses several fields of variable length to reduce, as much as possible, the data size of each discovered device to maximise the number of devices within an eHDDP message. Table 1 summarises the different values of MAC, ID and MTU sizes that have different wireless technologies and the typical maximum number of hops that they usually have. The last column shows, considering previous values, the maximum number of devices that an eHDDP message can hold, which is enough for most of the cases. Hence, the proposed variable length of fields allows the support of technologies usually used in IoT and WSNs such as LoRA, ZigBee or Z-Wave.

3.2. eHDDP operation in wired interfaces

This section describes the behaviour of eHDDP for wired interfaces, which corresponds with that of HDDP. We consider the wired topology shown in Fig. 2 to graphically explain the whole process. The hybrid topology example is composed by three SDN devices ($S1$, $S5$ and $S6$), three non-SDN devices (e.g., gateways $S2$, $S3$ and $S4$), and three non-SDN end devices (e.g., hosts $E1$, $E2$ and $E3$). Fig. 2(a) illustrates the exploration process triggered by the SDN controller. The controller creates a *Request* message with the *Num Device* equal to one, the *Device ID* field to the Data Path ID (DPID) of each SDN device, and finally, the *In Port* and *Out Port* fields to zero. Then, it sends a *Packet-Out* message, encapsulating the *Request*, to each SDN device in the domain (step 1) and commands them to flood the *Request* through all its interfaces.

When the *Packet-Out* message is received by an SDN device (S1, S5 and S6), it de-encapsulates and broadcasts the *Request* message as commanded (step 2). Then, as part of the confirmation phase illustrated in Fig. 2(b), it forwards the *Request* message back to the controller via a *Packet-In* message, similarly to OFDP (step 3). The controller checks the *Num Device* field to ascertain the number of non-SDN devices. If the number of devices is one, it means that the sending SDN device is a direct neighbour and no further action is needed. However, if the number of devices is greater than one, meaning the *Request* has traversed across non-SDN devices, further actions are needed to discover the topological information from them. Hence, the controller answers with a *Reply* message in a *Packet-Out*.

This *Reply* message is designed to collect the topological information from non-SDN devices and, guarantees all links between SDN and non-SDN devices are discovered (the process is later explained as steps 8–9). Going back to Fig. 2(a), the non-SDN devices cannot establish direct communication with the SDN control plane. Therefore, they must indirectly provide the topological information by embedding it in the control messages that will be later processed by the control plane. This process works as follows: *Request* messages received at non-SDN devices are simply broadcast via all their interfaces except the ingress one after increasing by one unit the *Num Device* field until an SDN device is reached (steps 3 and 4 in Fig. 2(a)). Furthermore, non-SDN devices must implement a locking mechanism similar to the one proposed in [26] to avoid loops and packet storms. As a result of this

Table 1
Max. number of devices in an *eHDDP* message.

Technology	MAC (Bytes)	ID (Bytes)	MTU (Bytes)	Typ. max # of hops	eHDDP max # of devs
WiFi	6	8	1500	20	**64**
	6	8	2304	20	**99**
LoRa [23,24]	4	4	255	5	**26**
ZigBee [25]	2	2	128	10	**16**
	8	8	128	10	**8**
Z-Wave [25]	2	1	64	4	**8**
	8	6	64	4	**4**

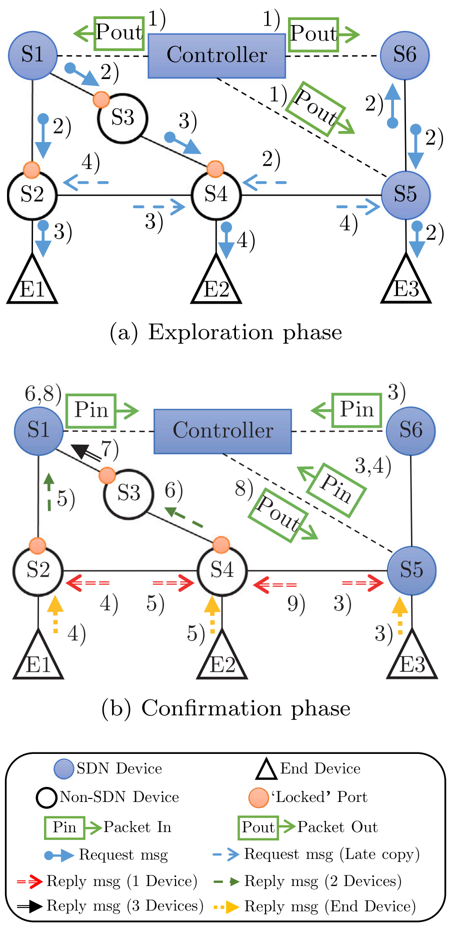


Fig. 2. Example of eHDDP in wired interfaces. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

process, late copies of the *Request* message (shown as blue dashed lines in the figure) arrive at other interfaces of non-SDN devices and trigger the sending of *Reply* messages backwards (steps 4 and 5 in red dashed arrows). A *Reply* message is built from the received *Request* message by changing the *Option Code*, setting the *Num Device* data to one and adding a new *Device Group* field to convey the required topological information.

Non-SDN devices that receive a *Reply* message increase the *Num Device* field by one and insert their own *Group Device* field. In addition, the *In-Port* field is set to the incoming interface of the *Reply* message and the *Out-Port* field is set to the locked interface which is the input interface of the first received *Request* message at that device (steps 5, 6, and 7) to direct the message towards the controller. Then, the newly conformed *Reply* message is sent through the interface specified in the *Out-Port* field. When a *Reply* message arrives to an SDN device, it is simply forwarded via a *Packet-In* message to the SDN controller, which collects all the gathered information across the non-SDN devices. Non-SDN end devices simply send a *Reply* message in response to a *Request* via their only interface (yellow dashed lines).

Finally, the controller processes the data conveyed in the *Reply* messages to build the full hybrid SDN topology.

3.3. *eHDDP operation in wireless interfaces*

In case of wireless or mixed (both wired and wireless) networks, such as SDWSNs, there are two main differences to consider regarding the wired operation in wired interfaces: (1) the locking mechanism, designed to prevent loops, cannot be applied to a port or interface since sensors usually have a single wireless interface, and (2) the number of devices in a wireless scenario is unknown, especially if device mobility is considered. These differences come from the fact that a wired interface defines the existence of a neighbour, but this fact does not hold in wireless networks in which a wireless interface can provide connectivity to many devices or none of them. This issue is especially relevant in most wireless technologies since there are usually based on connection-less communications over a shared physical layer.

The locking mechanism can be adapted to work on the identifier of the device that first sends a *Request* message instead of locking the incoming port. Wireless devices can be identified by their MAC address, which is embedded in the wireless frames. Hence, the locking identifiers in wireless networks are the MAC addresses from neighbour devices.

Regarding the second difference, there are two possible solutions to discover existing neighbours attached to a wireless interface: (1) to establish a proactive mechanism in which wireless devices periodically broadcast Hello messages and then inform the controller, which requires certain state depending on the number of neighbours and an additional packet exchange. Moreover, their use may increase the energy consumption (especially if the devices must be woken up for this purpose). (2) to establish a reactive mechanism in which wireless devices can discover their neighbours by following a request/response scheme in which any eHDDP message being broadcast acts like a request and reply messages act as responses. Moreover, a triple handshake *Request/Reply/ACK* is needed to detect potential unidirectional links from both sides of two direct neighbours. In such case, a new *ACK* control message is necessary, but it removes the need of storing the discovered neighbours to its later reporting and avoids unnecessary network activity in other instants of time with respect to the proactive mechanism. Both advantages reduce the energy consumption; hence, the second option is preferred in eHDDP since energy consumption is a key point in WSN and IoT networks.

We name the previously explained triple handshake exchange of messages as *eHDDP message exchange for bidirectional links*. The first two messages allow the discovery of a bidirectional wireless link to the device initiating the triple-handshake whereas the last two messages (the second message has a double use) allow the discovering of the same bidirectional wireless link to the neighbour device. However, if a device sending a *Reply* message does not receive the expected response to confirm a bidirectional link, it broadcasts the original *Reply* message to reach any existing neighbour, even across a unidirectional link. In

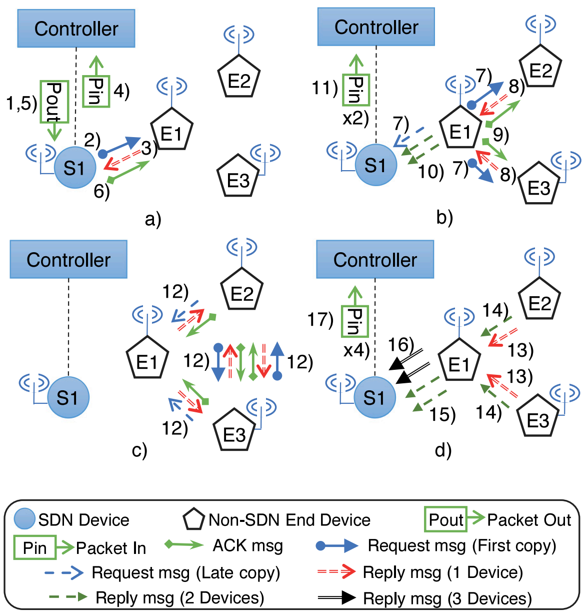


Fig. 3. eHDDP operation with only wireless bidirectional links.

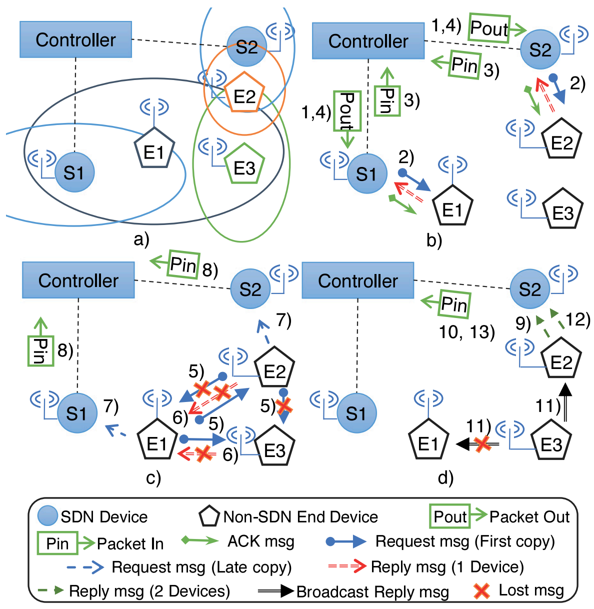


Fig. 4. eHDDP operation with some wireless unidirectional links.

this way, the topological information would be transmitted by the receiver to the control plane if possible. It is important to highlight that an *ACK* message must only be sent when a *Reply* message is received from a direct neighbour.

The following sections provide a detailed explanation of the protocol operation using two different case examples, first, a wireless network made of full bidirectional links, then, a second example mixing unidirectional and bidirectional wireless links.

3.3.1. Fully bidirectional links scenario

Fig. 3 illustrates an example with a single SDN device ($S1$) and three non-SDN wireless sensors ($E1$, $E2$, and $E3$), to clarify how eHDDP operates in wireless scenarios with only bidirectional links.

As depicted in Fig. 3(a), the SDN controller starts the discovery process by sending a *Request* encapsulated in a *Packet-Out* (step 1). $S1$ de-encapsulates the *Request* message and broadcasts it through its wireless interface (step 2). When the *Request* message is received by $E1$, it answers with a *Reply* message, as explained in Section 3.2 (step 3) that conveys the information of $E1$. Then, $S1$ relays the *Reply* message to the controller via the corresponding *Packet-In* (step 4). At this moment, both the controller and $S1$ know that the link connecting $S1$ and $E1$ is bidirectional, but $E1$ only knows that the link is working in the direction from $S1$ to $E1$, so a new message is needed to acknowledge the reception of the *Reply* at $S1$, thus confirming a bidirectional link between them. To this end, the controller sends an *ACK* message to $S1$ in another *Packet-Out* (step 5), which is de-encapsulated by $S1$ and sent to $E1$, thus completing the triple handshake (step 6). Now, the controller knows about $E1$ and the link connecting $E1$ and $S1$. If $E1$ did not receive the *ACK* message, it would mark the link as unidirectional and proceed as explained in the next Section 3.3.2.

The topology exploration process continues in Fig. 3(b). Upon reception of the *ACK* message $E1$ broadcasts the *Request* message (step 7). Both $E2$ and $E3$ receive the *Request* and answer with a *Reply* message (step 8). At this point, also $S1$ receives a copy of the *Request*, but it is discarded as a late copy of the original. Being $E1$ a non-SDN sensor, it directly replies with an *ACK* message (step 9) to both $E2$ and $E3$ to confirm the bidirectionality of the links. Then, $E1$ updates each *Reply* message with its own information and forwards them to the controller via $S1$ (step 10), which encapsulates every *Reply* in a *Packet-In* sent to the controller (step 11). Now, the controller knows about $E2$ and $E3$ and the links between them and $E1$.

$E2$ and $E3$ must also broadcast the *Request* message after sending the *Reply* as shown in Fig. 3(c). This fact implies four different *eHDDP message exchanges for bidirectional links* among $E2-E3$ (twice in opposite directions), $E2-E1$ and $E3-E1$ (step 12). The exchanges between $E2-E1$ and $E3-E1$ exist because the exploration process (*Request* messages) is based on broadcast.

Finally, Fig. 3(d) shows how $E2$ and $E3$ receive the *Reply* from $E1$, include their own *Device Group* tuple and forward the updated *Reply* to the control plane via $E1$ (step 13). On the other hand, the figure also shows how $E2$ and $E3$ update the *Reply* message which comes from each other and forward the updated messages to the control plane via $E1$ (step 14). Then, $E1$ updates the four *Reply* messages from $E2$ and $E3$ and sends them to $S1$. Now, two of the messages contain the information from two devices (step 15) and the other two messages contain the information from three devices, i.e., the loop $E1-E2-E3$ in both directions (step 16). Finally, $S1$ relays the four *Reply* messages to the control plane via *Packet-In* messages (step 17).

3.3.2. Mixed uni/bidirectional links scenario

Fig. 4 illustrates an example with two SDN devices ($S1$ and $S2$) and three non-SDN sensors ($E1$, $E2$ and $E3$) to clarify eHDDP operation in wireless scenarios with unidirectional links. It illustrates what happens when there are unidirectional links from $E1$ to $E2$, from $E1$ to $E3$ and from $E3$ to $E2$ as shown in Fig. 4(a).

Fig. 4(b) shows how the controller starts the exploration process sending as many *aceHDDP Request* via *Packet-Out* messages as many SDN devices exist (step 1). Then, $S1$ broadcast the *Request*, which is received at $E1$, thus triggering the triple handshake between them as previously explained (*Request/Reply/ACK*), note that the *ACK* is generated at the controller and relayed via a *Packet-Out* to $S1$ (steps 2, 3 & 4). Exactly, the same exchange occurs between $S2$ and $E2$.

When the previous exchanges finish, the process represented in Fig. 4(c) starts. Firstly, both $E1$ and $E2$ broadcast the *Request* message received from $S1$ and $S2$ respectively. Unfortunately, due to differences in range, only the *Request* message from $E1$ reaches $E2$ and $E3$, while the messages from $E2$ to $E1$ and from $E2$ to $E3$ get lost (step 5). Hence, the *Request* messages received at $E2$ and $E3$ from $E1$ trigger a *Reply* message that cannot reach back $E1$ because of the lack of range in that directions, i.e., they are unidirectional links (step 6). Moreover, the *Request* messages broadcast from $E1$ and $E2$ also reach $S1$ and $S2$ respectively (step 7) and both messages are sent to the controller via *Packet-In* messages (step 8) that will discard them. After sending a *Reply* each sensor starts a timer that is cancelled when the corresponding *ACK* is received. Fig. 4(d) illustrates what happens at $E2$ and $E3$ after the timer expiration since the *ACK* messages are not received due to lack of range in the directions from $E2$ to $E3$ and from $E3$ to $E1$. On the one hand, since $E2$ is able to reach the controller via $S2$, it includes the information regarding the unidirectional link from $E1$ to $E2$ and its own topological information in the *Reply* and sends it to the controller via $S2$ (step 9) and $S2$ encapsulates the message in a *Packet-In* (step 10). Now, the controller knows about the unidirectional link between $E1$ and $E2$. On the other hand, $E3$ does not know how to reach the controller, so it includes the information regarding the unidirectional link $E1$-$E3$ in its *Reply* and sends it in broadcast to inform about the unidirectional link to any potential neighbour (step 11) but it only reaches $E2$. Again, $E2$ knows how to reach the controller via $S2$, so it includes the information regarding the unidirectional link from $E3$ to $E2$ and its own topological information in the *Reply* and sends in unicast to the controller via $S2$ (step 12) and, again, and $S2$ encapsulates a *Reply* message in a *Packet-In* (step 13). Now, the controller also knows about both unidirectional links from $E1$ to $E3$ and from $E3$ to $E2$.

4. eHDDP logic and implementation details

The logic of eHDDP has been divided into two different parts derived from its use in hybrid SDN networks. The first part defines an eHDDP software agent that implements the protocol functionality in non-SDN devices while the second part defines the behaviour of the controller application, which is in charge of processing the eHDDP control messages received at SDN devices.

4.1. eHDDP agent for non-SDN devices

The eHDDP agent follows a different logic depending on the type of interface (wired or wireless as shown in Fig. 5) receiving the protocol messages: Fig. 5(a) presents the logic followed by the eHDDP agent when the control messages arrive from a wired interface, and Fig. 5(b) when they come from a wireless one. Hence, the first step in any eHDDP agent is to check the type of the incoming interface to apply the appropriated wired or wireless control logic. It is important to highlight that both wired and wireless interfaces implement a locking mechanism based on source MAC address to prevent loops and simplify the coding of both modes. Moreover, if the device has more than one interface, the eHDDP agent must also store the incoming interface associated with the locked source MAC address.

The implementation of eHDDP is based on Linux raw sockets and is coded in Python. This election is chosen for simplicity since it allows a full development in the Linux user space. Moreover, it can be easily deployed in the testbed platform described in Section 5.

4.1.1. Processing logic for wired interfaces

Fig. 5(a) summarises the logic applied by the eHDDP agent if a control message comes through a wired interface.

On the one hand, if the incoming control message is a *Request* message, the agent checks if there is a previously locked source MAC. If it is the first *Request* message received and the device has more than one interface, it locks the source MAC, then increases the value of the *Num Device* field by one and broadcast the message. Otherwise, if the source MAC is already locked or the device has only one interface, a *Reply* message is created in response following the structure represented in Fig. 1 and sent through the incoming interface of the *Request* message.

On the other hand, if the incoming control message is a *Reply* message, the device increases the value of the *Num Device* field by one and inserts a new *Device Group* to include the link traversed at the end of the control message. Finally, the *Reply* message is forwarded through the interface associated to the locked source MAC address. Otherwise, it is simply discarded.

4.1.2. Processing logic for wireless interfaces

Fig. 5(b) summarises the logic applied by the eHDDP agent if a control message comes through a wireless interface.

On the one hand, if it is a *Request* message, the agent processes the packet increasing by one the *Num Device* field and changing the value of *previous MAC* field for its wireless MAC, and then saves the message if the source MAC address has not yet been locked which happens when the first *ACK* message is received guaranteeing a bidirectional path to the controller. Additionally, the agent creates a *Reply* message including its *Device Group* tuple and sends it through the incoming interface of the *Request* message. Moreover, the agent starts a timer to wait for the corresponding *ACK* message, thus confirming the link works in both directions.

If the *ACK* message is not received because of a packet loss, the timer expires and the *Reply timeout* triggers the retransmission of the *Reply* message. The *Reply* message can be retransmitted up to n times. Finally, if no *ACK* is received after the nth retransmission, the *Reply* retransmission attempt is finally broadcast to look for potential devices with a known path to the controller.

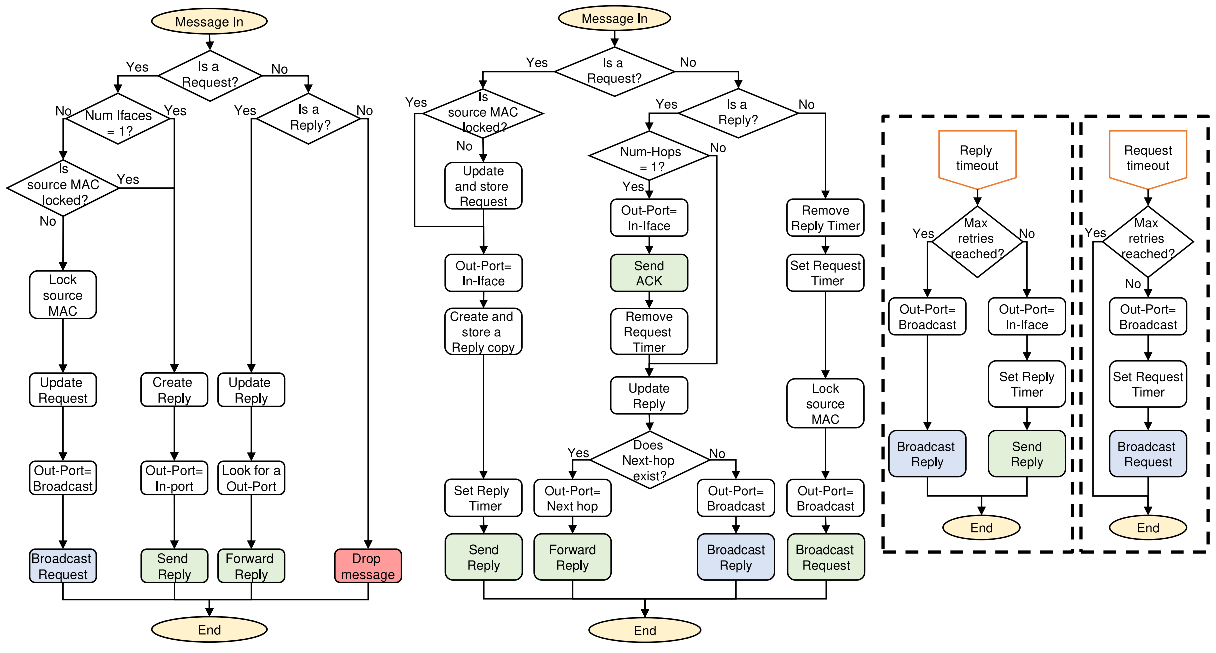
On the other hand, if it is a *Reply* message, the agent creates and sends back an *ACK* message if the original *Request* came from a direct neighbour (i.e., the value of the *Num Device* field is one). Moreover, regardless of the value of the *Num Device* field, the agent updates the *Reply* message by increasing the *Num Device* by one, replacing the value of *Previous MAC* address with its own MAC address and inserting a new *Device Group* with the information for the new link (Fig. 1). Later, the agent checks if an outgoing interface exists to reach the controller. In such case, the updated *Reply* message is sent via that interface. Otherwise, the device broadcasts the *Reply* message so that it reaches any potential neighbour with a known path to the controller.

Otherwise, when an *ACK* message arrives, the agent stops its associated timer, locks the source MAC address of this message as next hop towards the controller, and broadcasts the original *Request* message linked to the received message. Moreover, the agent starts a timer to wait for *Reply* messages from neighbour devices.

If no *Reply* message is received because of packet losses, the timer expires and a *Request timeout* event is triggered. The *Request* message can be retransmitted up to n times to mitigate the effect of possible packet losses.

4.1.3. Applicability of eHDDP in low-powered devices

There are wireless devices with important hardware and software limitations such as IoT or WSN devices; thus, eHDDP is designed to use the minimum number of resources in wireless devices. On the one hand, eHDDP only requires storing the first received *Request* message and its input interface to later forward the required *Reply* messages towards the controller or to retry "n" times the broadcast of the *Request* message if no *Reply* messages are received due to packet losses. On the other hand, it is also necessary to temporarily store the *Reply* messages generated by a device as response to *Request* message until an *ACK* is received or "n" retries of sending the message are performed. After the nth retry, if the broadcast timer expires again, the *Reply* message is sent in broadcast and it can be safely removed from the device. The number of retries may be configured depending on the packet error probability or the Signal-to-Noise Ratio (SNR). See Section 6.3 for further details.



(a) Logic for wired interfaces (b) Logic for wireless interfaces

Fig. 5. eHDDP agent.

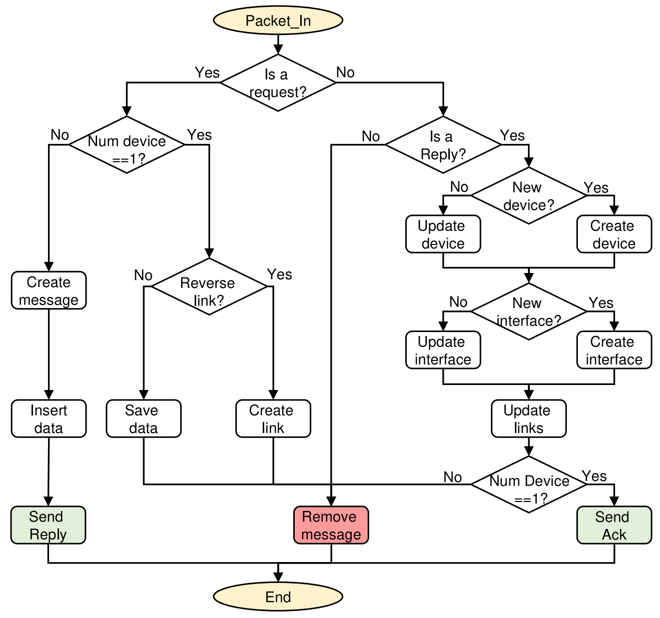


Fig. 6. eHDDP controller application logic.

4.2. *eHDDP controller application*

The SDN control plane must implement the logic summarised in Fig. 6 to handle the topological information conveyed by eHDDP control messages from the data plane. This logic has been implemented in an Open Network Operating System (ONOS) controller application and uses the OpenFlow protocol for SDN device-controller communication. Accordingly, the next paragraphs rely on an OpenFlow agent installed at SDN devices capable of handling *Packet-In* and *Packet-Out* message exchanges with the controller to better illustrate the behaviour.

The controller is in charge of starting the exploration process by sending as many *Request* messages via *Packet-Out* as SDN devices were previously discovered by the OpenFlow agent.

If the controller receives a *Request* message and the value of the *Num Device* field is greater than one, it answers with a *Reply* message with the corresponding *Device Group* info via a *Packet-In*. Otherwise, if the value of the *Num Device* field is one, the controller stores the information until a *Request* message in the opposite direction is received to create the link between both devices. If the controller receives a *Reply* message, it extracts the information from the message (devices and links), saves it in its topological database and, if the *Num Device* field has a value of one, replies with the corresponding *ACK*. Finally, in case the controller receives an *ACK* message, it simply discards it.

4.3. *Open source code available on GitHub*

The implementation of eHDDP for wired devices (based on the BOFUSS software switch [27]) and wireless devices (python code) as well as the controller application developed for the ONOS controller [28] are available as open source in a GitHub repository [29]. We encourage any reader interested in the topic to test the implementations and provide any feedback.

5. Testbed

The testbed for the evaluation of eHDDP is also an important contribution to the work presented in this paper since there was not any tool fulfilling all necessary requirements. Selecting an appropriated platform is not a trivial task since it is necessary to evaluate full hybrid SDN scenarios with both SDN and non-SDN devices in the same topology, mixing also wired and wireless technologies. These constraints limit the available tools to carry out the evaluation tests.

5.1. *Evaluation platform*

There are two possible options to evaluate networking protocols and architectures different from implementing them in real hardware: network simulators and emulators. The choice of one or the other will depend on the required development time speed, the versatility to set up different scenarios or their capability of modelling real environments. Network simulators present high scalability but are slightly far from real environments and consume much time to carry out the simulations, whereas network emulators offer scenarios and deployment closer to real environments at the cost of requiring more computing resources.

The first category (network simulators) has some alternatives with good support for wired and wireless scenarios such as ns-3 simulator [30], though the simulation models for SDN networks in these platforms are usually limited. Although there are simulators specially focused on WSNs or SDWSNs that allow to evaluate wireless scenarios and set up complex topologies, they do not support wired communications. An example is Cooja [31] under the Contiki Network Simulator [32] that emulates low-power wireless scenarios with sensors that run under the Contiki-ng [33] operating system.

Within the network emulators category, the most widely used tool is the open sourced Mininet [34] emulator which is coded in Python and runs on Linux based systems. It is based on Open virtual Switch (OvS) [35] and Basic OpenFlow Software Switch (BOFUSS) [27] virtual software switches and also on virtual hosts or devices based on Linux *name spaces*, which are used in our implementation of the eHDDP agents. However, it lacks direct support for wireless interfaces which is provided by additional project forks such as: Mininet-WiFi [36] for WiFi [37] interfaces, and Mininet-IoT [38] for IEEE 802.15.4 [39] wireless interfaces commonly supported by IoT devices. The key point is the fact that both of them allow devices with wired and/or wireless interfaces. Mininet-WiFi is selected for the experiments because it is a repository in continuous update and better maintenance than Mininet-IoT (both of them are the from the same author). If necessary, the proposed changes to Mininet-WiFi can be ported to Mininet-IoT.

5.2. Mininet-WiFi enhancements for Ad-Hoc networks

Although Mininet-WiFi is an interesting tool, it cannot be used as it is for the study of complex WSNs or SDWSNs since they usually work in Ad-Hoc mode, while Mininet-WiFi is mainly focused on providing an emulation environment for the Infrastructure mode in WiFi networks [40] based on Access Points (APs). This limitation comes from the fact that Mininet-WiFi can only define signal ranges between pairs of nodes, which is implemented between an Access Point (AP) and a network device or in Ad-Hoc wireless networks with only two devices. Hence, two new functionalities enhance the support of Ad-Hoc networks in Mininet-WiFi: a new Ad-Hoc multi-node class and the addition of a wrapper to the existing mobility module capable of discarding packets between devices out of range according to the position and signal ranges setup by the Ad-Hoc multi-node class.

The new Ad-Hoc multi-node class allows the association of any number of end-devices to a wireless Ad-Hoc network by Mininet-WiFi. Furthermore, it allows to individually define the signal range for each device. The new functionality of the mobility module is in charge of deciding whose devices are in range and the new wrapper, which is based on eXpress Data Path (XDP) [41], is in charge of discarding packets received from out-of-range devices. First, the new functionality of the mobility module obtains the Euclidean distance between two devices. Then, the mobility module compares the distance obtained with the maximum range of those devices. If the distance between them is less than their maximum range, the module updates the XDP program to discard the packets out of range from the transmitter. This operation is performed for all the devices of the network and each time that any device changes its position. Hence, when a packet arrives at a device, its XDP wrapper reads the source MAC/ID and discards the packet if it is out of range according to the information previously provided by the mobility module.

XDP technology is selected because it is the first entry point in the Linux kernel to filter packets just after receiving them from a network interface and before its delivery to any of the socket families in the Linux kernel, including the Linux raw sockets used in our implementation of the eHDDP agent for non-SDN devices. Hence, this way of filtering allows the emulation of a packet loss in a wireless medium since the packets are discarded previously to their processing in the TCP/IP Linux stack, which assures that the packets are never processed by the devices as desired.

5.3. Hardware infrastructure

The customised version of Mininet-WiFi and the ONOS SDN controller run on two machines powered by Intel(R) Core(TM) i7 processors with 16 GB of RAM to perform all the experiments in the evaluation section.

5.4. eHDDP setup

The evaluation platform is based on Mininet-WiFi, which defines the IEEE 802.11 standard as the link layer in the performed experiments. Mininet-WiFi is used with its default setup in which there is a one hop out-of-band control channel with the control plane: an ONOS controller running the eHDDP controller application logic. The timer to trigger retries for *Request* and *Reply* messages is set to a conservative value of 300 ms to minimise the effect of packet errors and avoid the triggering of unnecessary broadcast *Reply* messages. This value may be adjusted according to the wireless technology in use. The experiments are performed without background traffic emulating the fact that eHDDP messages are transmitted through the highest priority queue in the network devices. This fact would be equivalent to use 802.11e QoS extensions for the mentioned purpose.

5.5. Measurements

The metrics selected to evaluate the performance of eHDDP are the number of exchanged packets, the discovery time, the controller processing time and the maximum number of stored messages during an exploration interval. The number of exchanged packets is a measurement that represents the overload introduced in the network by eHDDP and gives an overview of its scalability. The discovery time measures the time required by eHDDP to convey the topological information, embedded in the *Reply* messages, to the control plane. This measurement is a key factor in wireless networks since it defines how changes in a network topology can be detected in case of device mobility. The controller processing time measures the time consuming by the SDN controller to build the topology taken into account the topological information collected by eHDDP. The controller processing time always will be higher than the discovery time, since it includes discovery time. The results obtained from all SDN device topologies are compared with those obtained by OFDP [42] because it is the default mechanism implemented by the selected ONOS version in the experimental setup. Finally, the maximum number of stored messages represents the required state to support eHDDP.

Measurements in lossy wireless hybrid SDN network topologies are conducted considering packet losses of 0, 0.5, 1 and 2% at all wireless interfaces. Note that the packet losses are configured at the eHDDP level rather than at the PHY level because Mininet-WiFi (the emulation platform in use) relies on a Linux TC command to generate random packet losses in ad-hoc scenarios [43]. This setup produces losses at the transmitter side, thus ignoring the MAC retransmission procedure. As a result, the corresponding packet losses at the PHY level are much higher (for instance, 2% packet losses at the eHDDP level corresponds to 57% at the PHY level for the case of seven failed MAC transmissions in a row). Moreover, by enforcing the packet losses at the transmitter side, we neglect the possibility to receive the packet at other nodes different from the one affected by the faulty link, thus increasing the effect of packet losses. We study the protocol performance setting the maximum number of retries for eHDDP Request and Reply messages to 0 (no retries), 1 and 2 retries. Finally, a conservative timeout of 300 ms

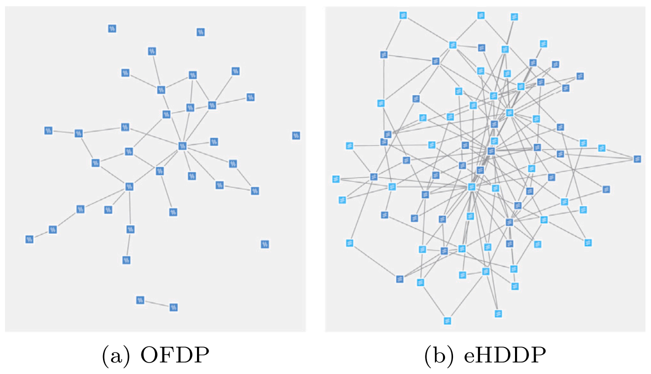


Fig. 7. Hybrid SDN Géant-like network topology. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

between retries is enforced to minimise the effect of neglecting the MAC retransmission procedure in the discovery time.

All measurements are calculated with at least ten runs of each experiment. Experiments with random topologies present 95% confidence intervals while regular topologies show standard deviations since they are deterministic.

6. Evaluation

This section evaluates the performance of eHDDP under different topologies. Nevertheless, we perform first a qualitative study based on the GÉANT pan-European network topology [44] (see Fig. 7) but randomly assigning the type of device (SDN or non-SDN) to each network node. Fig. 7(a) shows the topology obtained using OFDP while Fig. 7(b) shows the topology obtained using eHDDP. The SDN devices are represented in dark blue and non-SDN devices are in light blue. OFDP is unable to discover non-SDN devices which produces isolated SDN devices in the discovered topology; whereas eHDDP is capable of obtaining the full topology with all the existing devices. It can also be observed how the previously isolated SDN devices in dark blue are connected now to other SDN devices through non-SDN devices in light blue. Hence, the functionality of eHDDP is demonstrated in a well-known complex topology.

After the qualitative study, we carry out different quantitative experiments to measure the performance in wired SDN hybrid networks as well as on wireless SDN hybrid networks including unidirectional links in some cases. Finally, full-hybrid networks in which SDN and non-SDN devices can have wired or wireless interfaces are also considered.

6.1. Wired hybrid SDN network topologies

The evaluation of eHDDP in wired SDN hybrid networks topologies is based on random topology models (Barabási–Albert [45] and Waxman [46]) with different number of devices, connectivity degree and percentage of SDN devices. The aim is to check the behaviour of eHDDP in several scenarios of diverse nature.

Fig. 8 represents the number of exchanged messages per discovery process for Barabási–Albert topologies and Fig. 9 shows the same information for Waxman topologies. Both figures present the results for network topologies composed by 20, 40 and 80 devices and different average node degrees (2, 4 and 6). An important feature that can be easily observed in both figures is that the number of required messages (continuous red line with star markers) is close to a linear increase with the average degree of the network. These means that the number of messages is proportional to the number of links which is a good scalability indicator. The number of OpenFlow packets does not change significantly regarding the variation in the percentage of SDN devices. Any increment in the number of *Packet-In* messages when the percentage of SDN devices is high, entails a decrease in the number of *Packet-Out* messages. Hence, the number of messages that the control plane must process to obtain hybrid network topologies and the required processed load do not suffer of important changes if a network increases its ratio of SDN devices. Obviously, the total number of *eHDDP* messages is inversely proportional to the percentage of SDN devices, the more SDN devices they are, the less message exchanges are needed to discover the whole topology. This fact is confirmed by observing the decrease in the number of *Reply* messages as the percentage of SDN devices rises. However, that does not hold for the number of *Request* messages required. When the number of SDN devices is small, *Request* messages are mainly generated by non-SDN devices to explore the network, while in the opposite case, the number of *Request* messages comes from the control plane to initiate the exploration process from all SDN devices. In this case, the number of *Request* messages is very similar but smaller than the number of OFDP messages if all the devices are SDN because their operation procedure is almost the same in SDN network topologies. Indeed, if all the devices are SDN, the number of *Packet-Out* messages equals the number of *Request* messages and the are no *Reply* messages since there are no non-SDN devices.

The differences between Barabási–Albert and Waxman topologies are in the number of messages needed by eHDDP for the same set of topology parameters, Barabási–Albert topologies are created following a power-law model whereas Waxman follows a random model. These model variations provoke that, for the same set of topology parameters, Barabási–Albert topologies feature a smaller network diameter than Waxman topologies thus, they require a smaller number of exchanged messages to discover the full topology.

It is worth mentioning that there are *ACK* messages (as shown in Figs. 8 and 9). They are generated by our ONOS application because the OpenFlow specification cannot distinguish if an interface is wired or wireless. Hence, if a *Reply* message is received inside a Packet_In, it must answer with an *ACK* message inside a Packet_Out. The eHDDP agent discards the *ACK* message if it arrives at a wired interface or process it according to Sections 3.3 and 4.1.2 if it arrives at a wireless interface.

Fig. 10 shows two types of time measurements (discovery time and processing time) for both random topology models. The discovery time is the time elapsed since the controller sends the first *Packet-Out* message, until the reception of the last *Packet-In* message, whereas the processing time is the time taken by the controller to build the topology, it measures the time elapsed since the controller sends the first *Packet-Out* message until it has finished processing all the topology-discovery packets. Therefore, the processing time will always be higher than the discovery time. Again, as with the number of protocol messages, the discovery time in Barabási–Albert topologies is smaller than in Waxman topologies. The explanation is the same, discovery time in Barabási–Albert networks is smaller than in Waxman networks due to their smaller network diameter, and the processing time at the ONOS controller is higher than the discovery time independently of the topology shape. However, this last fact is especially relevant when the number of SDN devices is small, and negligible if most of the devices are SDN. This behaviour is motivated because SDN devices are previously added to the ONOS database by its OpenFlow agent but non-SDN devices must be added on the fly according to the topological information conveyed by the *Reply* messages, which increases the processing time. To conclude, both times show a proportional increase with respect to the number of devices and their degree, consuming more time when the more devices and links exist. This increase is aligned with the behaviour observed in the number of exchanged messages.

6.2. Lossless wireless hybrid SDN network topologies

This section presents the evaluation of eHDDP in lossless wireless hybrid SDN topologies. First, a qualitative study is performed to assess

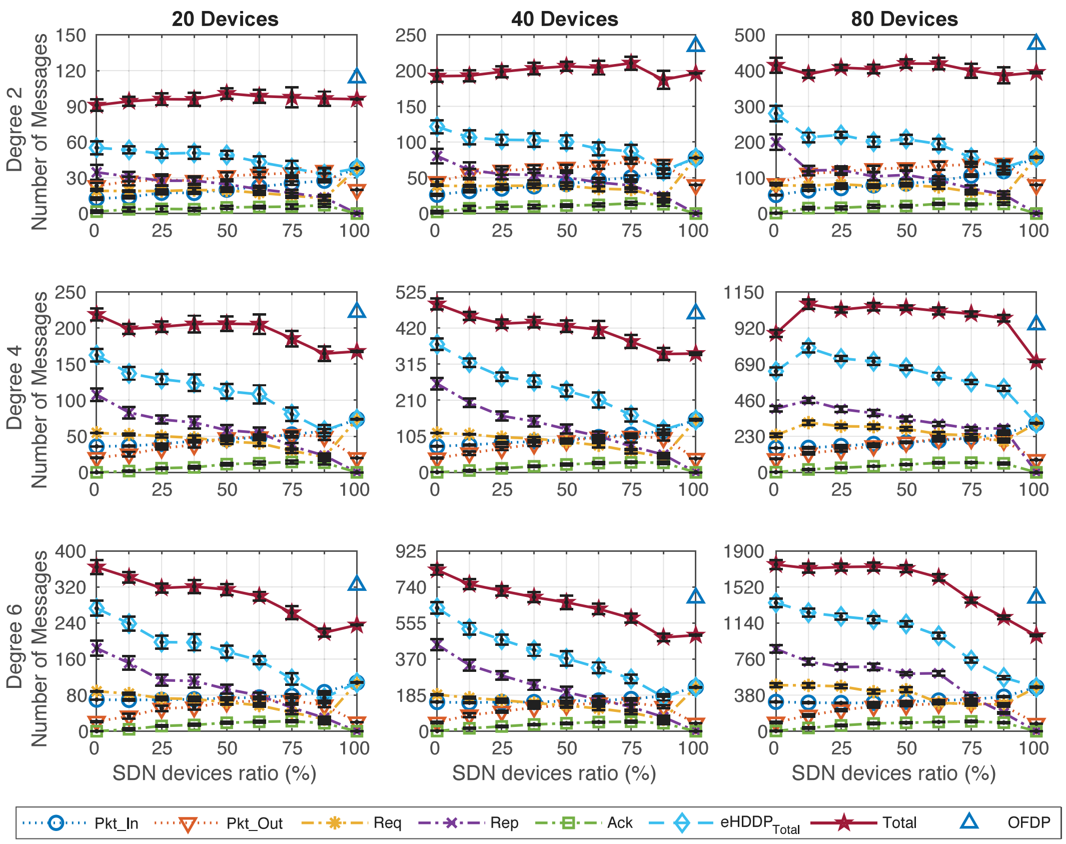


Fig. 8. Number of messages in Barabási–Albert topologies.

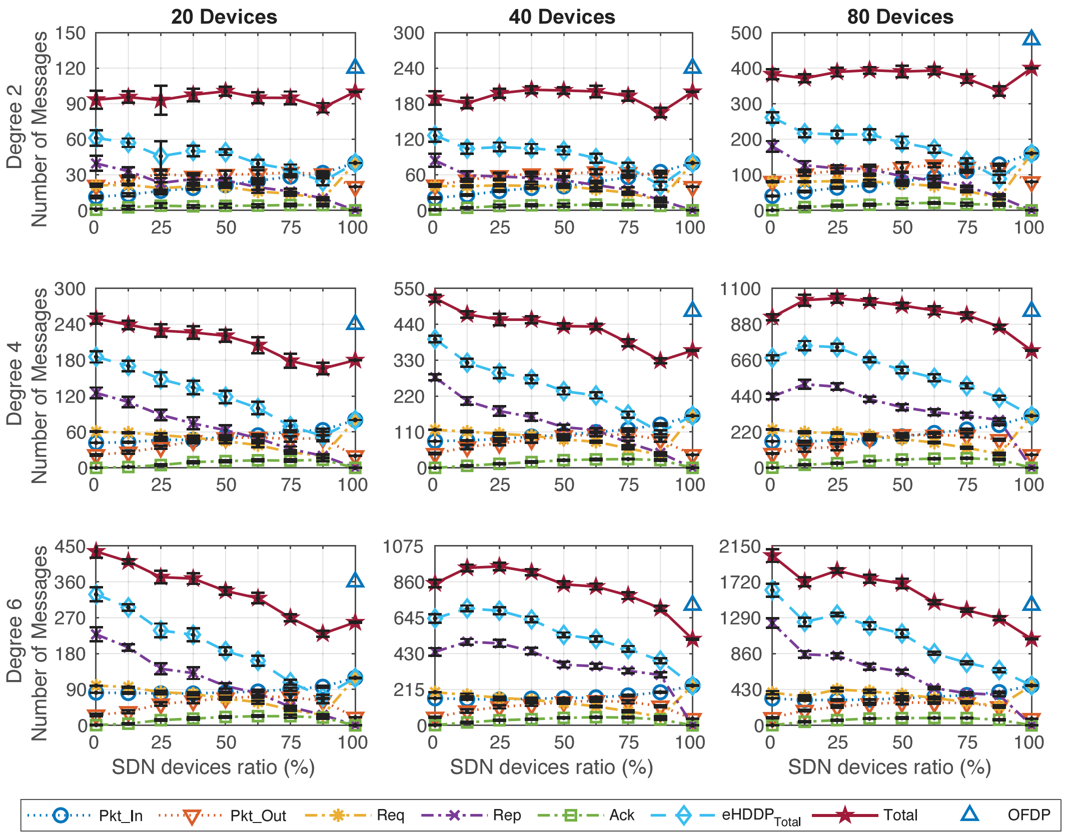


Fig. 9. Number of messages in Waxman topologies.

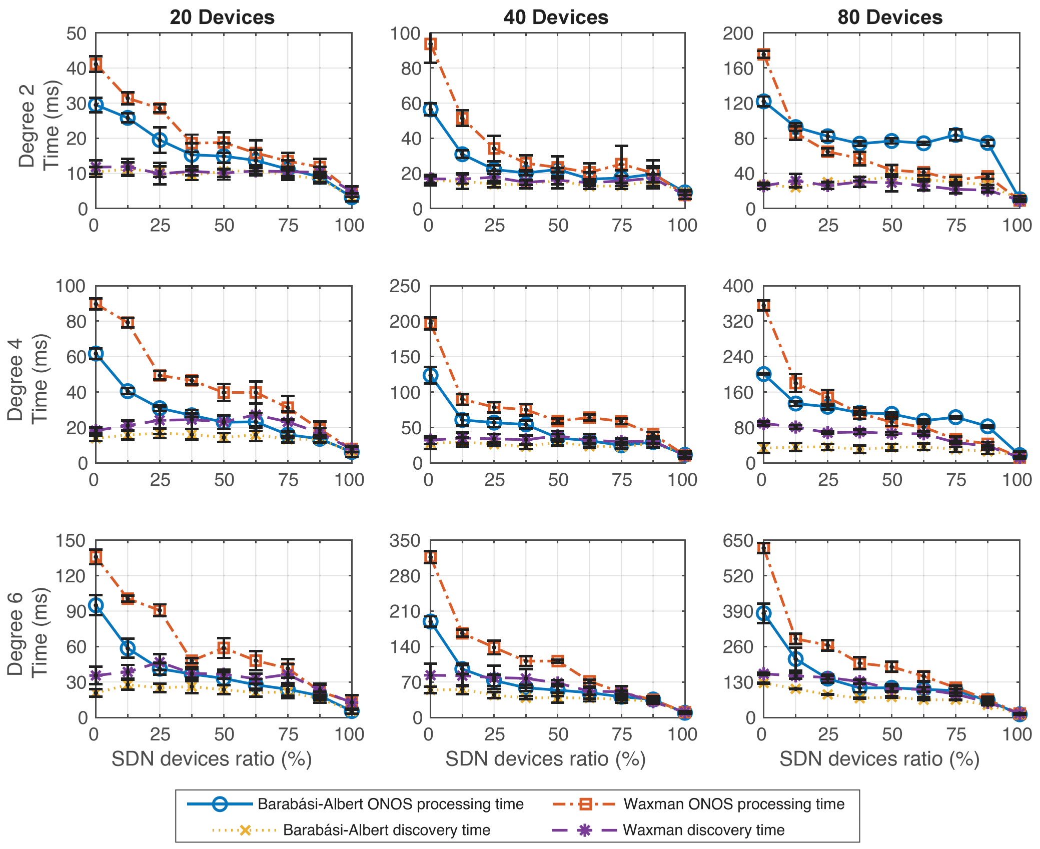
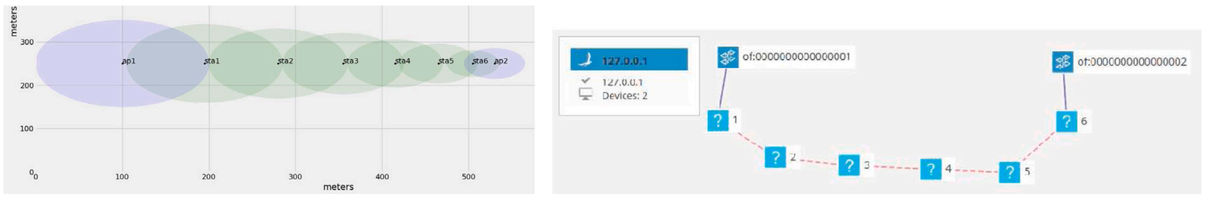


Fig. 10. Discovery.



(a) Mininet (b) ONOS

Fig. 11. Linear network topology with unidirectional links. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

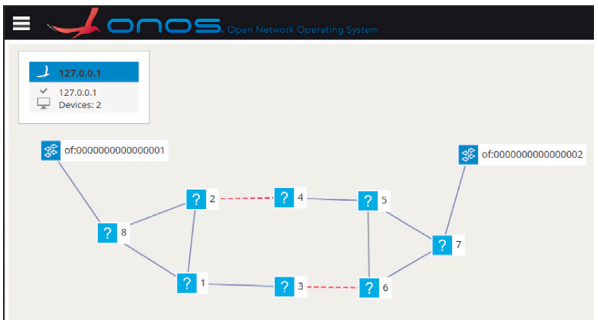
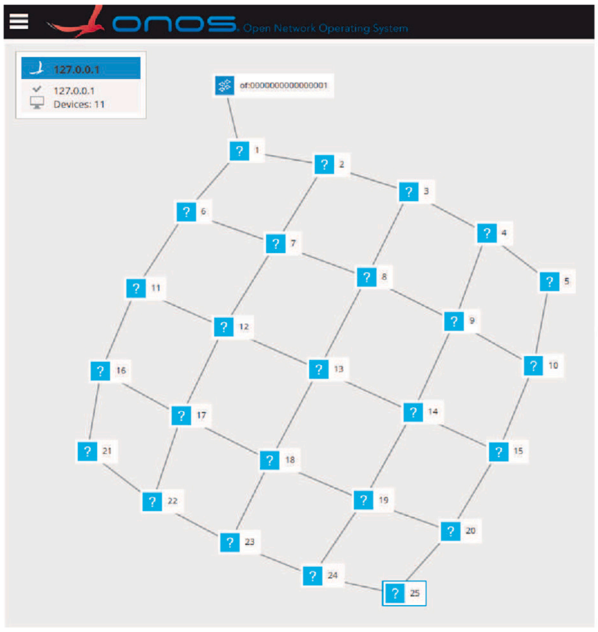


Fig. 12. WSN clusters connected by unidirectional links.

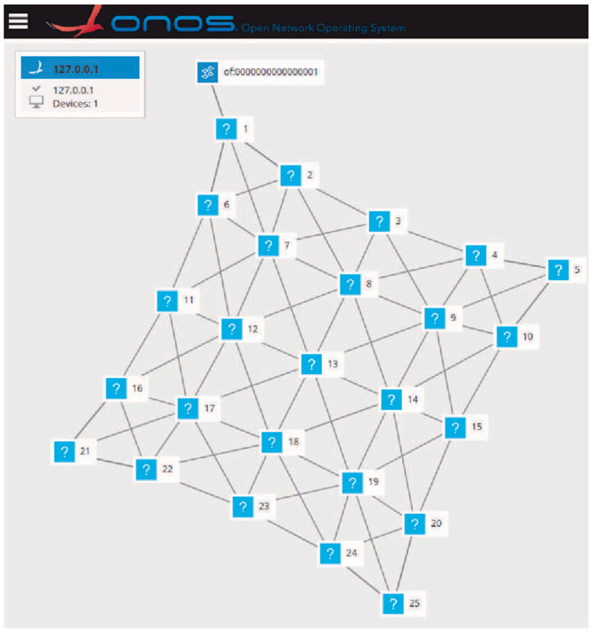
eHDDP can deal with unidirectional and bidirectional wireless links. Fig. 11 shows how Mininet-WiFi is configured to define a linear wireless topology made of unidirectional links (Fig. 11(a)) and the corresponding topology acquired at the ONOS controller by using eHDDP. The configured topology has two SDN devices (*ap1* and *ap2*) connected by six wireless devices conforming a unidirectional wireless path from *ap1* to *ap2*, which is built by chaining wireless devices with decreasing signal range. Dark blue boxes represent SDN devices and light blue boxes are wireless devices. ONOS depicts bidirectional links in continuous lines and unidirectional links in dashed lines. Unfortunately, ONOS cannot show the direction of unidirectional links, although this information is stored in its topology database.

On the other hand, Fig. 12 shows a wireless network composed by two small clusters connected through two unidirectional links with reverse directions. Although there are no direct bidirectional paths between both clusters, eHDDP is capable of discovering the whole topology thanks to the broadcast of *Reply* messages.

After the previous qualitative study, we present a quantitative study following the same structure used in the evaluation of eHDDP in wired network topologies, but in this case for wireless network topologies. In wireless networks, the connectivity between neighbours is determined by the geographical position and the signal range which constraints the definition of random topologies. We manually built squared meshes (see Fig. 13) with connectivity degree 4 (Fig. 13(a)), and connectivity degree 6 (Fig. 13(b)) by individually modifying device positions and signal ranges.



(a) Degree 4



(b) Degree 6

Fig. 13. Wireless 5 × 5 mesh topologies with one SDN device.

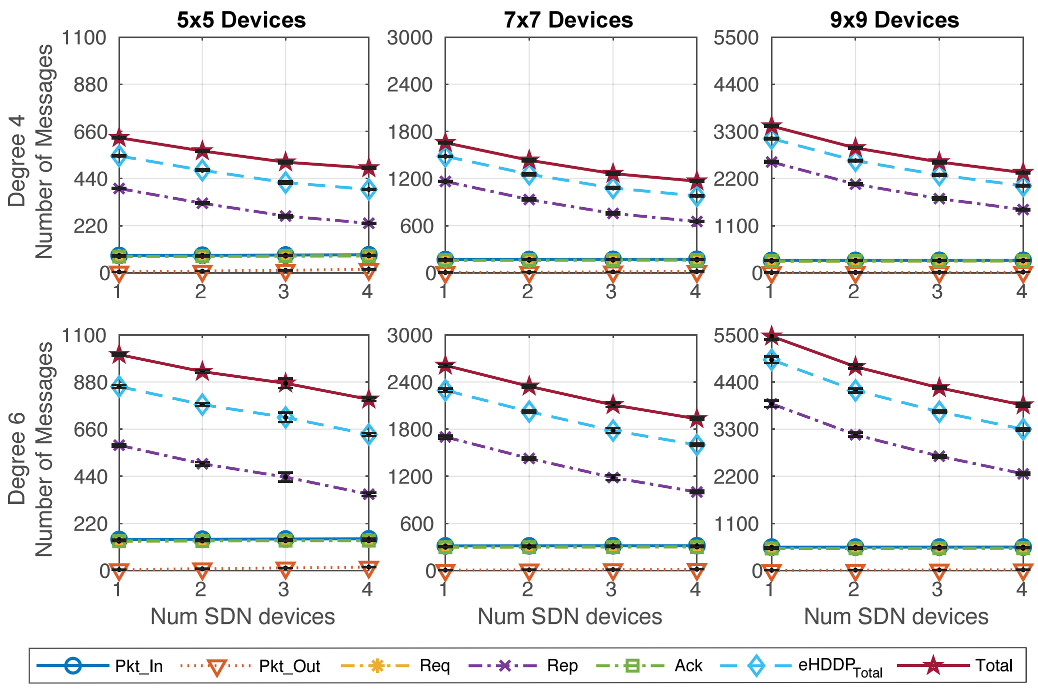


Fig. 14. Number of control messages in wireless topologies.

The results in terms of number of exchanged messages are collected in Fig. 14, and their discovery and processing times are summarised in Fig. 15. Both figures, show the experiments with 1, 2, 3 or 4 SDN devices, always placed at the corners of the mesh topologies. The number of exchanged messages shown in Fig. 14 follows the same behaviour than in wired topologies, growing linearly with the number of links. Additionally, the number of required messages decreases if the number of SDN devices increases, since the average distance with respect to an SDN device and the exploration processes require shorter paths to convey the information to the control plane.

Although, there are differences between the example topologies used for wired and wireless topologies, it is interesting to compare the results between them. We can compare the results from wired topologies with 80 devices and average connectivity degrees of 4 and 6, and wireless 9 × 9 squared mesh topologies (81 devices) and also average connectivity degrees of 4 and 6, the following aspects can be observed. Regarding the number of exchanged packets, wireless topologies doubles or triples the results from wired topologies due to the new *ACK* message used to confirm link bidirectionality and the need for some extra *Reply* messages on wireless devices in which a wireless interface may have several neighbours attached. Each wireless neighbour device that receives a *Reply* message tries to forward it towards the control plane for reliability which increases the number of required *Reply* messages.

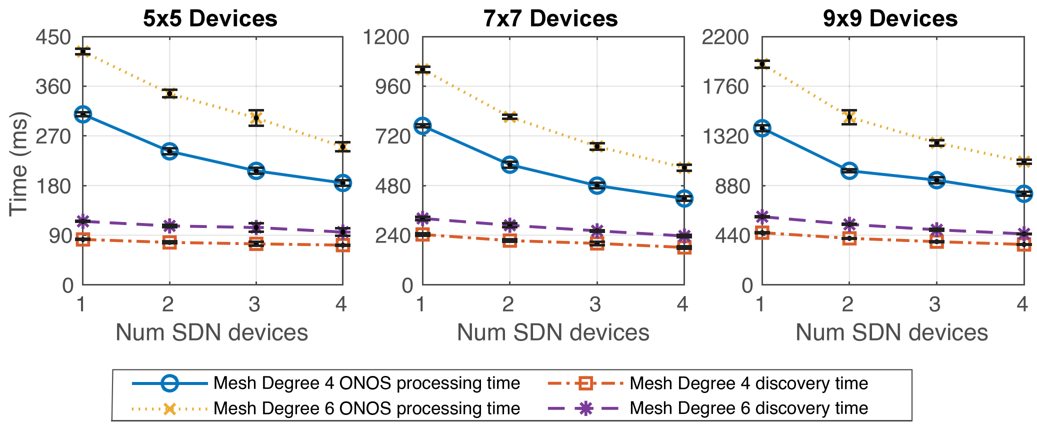


Fig. 15. Discovery and processing times in wireless mesh topologies.

Regarding discovery and processing times, Fig. 15 presents the results in wireless scenarios, showing a similar trend to those of the wired ones but the absolute values are far from close. The increase in discovery time derives from the very nature of wireless interfaces. They work in semi-duplex mode, so they are slower and less reliable than wired interfaces that work in full-duplex mode. The difference with respect to the wired topologies in the ONOS processing time is mainly due to the controller needs to process more messages. The Fig. 15 only shows the worst-case scenario which is when the number of SDN devices is small (from 1 to 4 devices). Note how the discovery time and especially the processing time clearly decrease with the addition of a single SDN device.

Finally, it is important to highlight that the measured discovery/processing times give an idea on how fast the control plane can launch contiguous eHDDP discoveries without overlapping in order to deal with the mobility of the wireless devices by refreshing the topological data.

6.3. Lossy wireless hybrid SDN network topologies

This section presents the evaluation of eHDDP in lossy wireless hybrid SDN topologies. The experiments are performed in an adverse scenario, wireless network topologies of degree 4 (lower connectivity that degree 6) and a single SDN device (there is a unique point to reach the SDN controller). Apart from the 7×7 square mesh used in the lossless scenario, we also consider random topologies with 49 wireless devices generated following the Leipzig model of the topology generator described in [47]. The single SDN device in the topology is randomly chosen in each experiment. Fig. 16 shows the results regarding the number of control messages, the network discovery time, the percentage of links discovered and the maximum number of stored messages in the devices during the experiments. The experiments are repeated to consider packet losses at 0, 0.5, 1 and 2% at the eHDDP level, as it was explained previously in Section 5.5. Moreover, the maximum number of Request and Reply retries is set to 0 (no retries), 1 and 2.

The number of messages exchanged in each discovery process decreases with packet error probability if no retries are used. The use of retransmissions in *Request* and *Reply* messages makes the number of transmitted control messages slightly higher with respect to the experiments without losses. This fact make sense since the retransmission overcome the packet losses but the lost packets are included in the performed measurements. Although it is a small increase on average, it exhibits high variability (the size of the confidence intervals) in Leipzig topologies due to their random nature and the random selection of the single SDN device. Topologies following the Leipzig model feature a well connected central core spanning several branches that connect devices almost in line, so choosing a device located in one of the branches as the SDN device has a serious impact on its performance. Obviously, there is no difference regarding the number of retries if there is no packet error probability.

Regarding the discovery time (the time required to convey all the topological information), it increases with the packet error probability and the number of retries. On the one hand, the packet error probability produces some *Reply* messages in broadcast, which increases the required number of messages and the discovery time as a consequence. On the other hand, it can be easily observed the effect of the conservative 300 ms timer used between retries in the discovery time. Its increase is proportional to the 300 ms retry timer and the number of retries in use. Again, the discovery time in Leipzig topologies is higher because they exhibit a higher network diameter.

The percentage of links discovered decreases as the packet error probability increases. Nevertheless, the percentage of discovered links remains high and over 94% for the worst-case scenario of 2% packet error probability.

Finally, the right column in Fig. 16 shows the average of the maximum number of control messages stored in memory during the experiments by any wireless device. We can observe how its value is small independently of the packet error probability and depends on the type of topology. Leipzig topologies require a larger number of packet in memory since their diameter is larger, thus increasing the average path to the controller and the total number of messages as shown on the left part of the figure. In any case, the required memory is small with respect to the number of exchanged messages.

We can conclude that there is a trade-off between the percentage of discovered links and the number of retries since both the number of exchanged messages and the discovery time increase with the retries. The timer between retries may be reduced but depends on the wireless technology in use, this fact is considered as future work. All in all, these results demonstrate that eHDDP is suitable for use even on lossy wireless hybrid SDN networks requiring a limited number of resources independently of the packet error rate.

6.4. Full-hybrid network topologies

This section assesses eHDDP in a full-hybrid network topology to conclude the evaluation of the proposal in a complex use-case scenario. Fig. 17 illustrates a topology composed by a wired backbone connecting two different wireless networks. This example is selected because the SDN control plane must control and manage the backbone network as well as the wireless networks. As in the previous section, Fig. 17(a) illustrates the topology under study built in Mininet and Fig. 17(b) the topology represented by ONOS after launching eHDDP. We see how eHDDP succeeds in discovering the whole topology, showing the link connections among all the devices and guaranteeing its functionality both in wired and wireless networks.

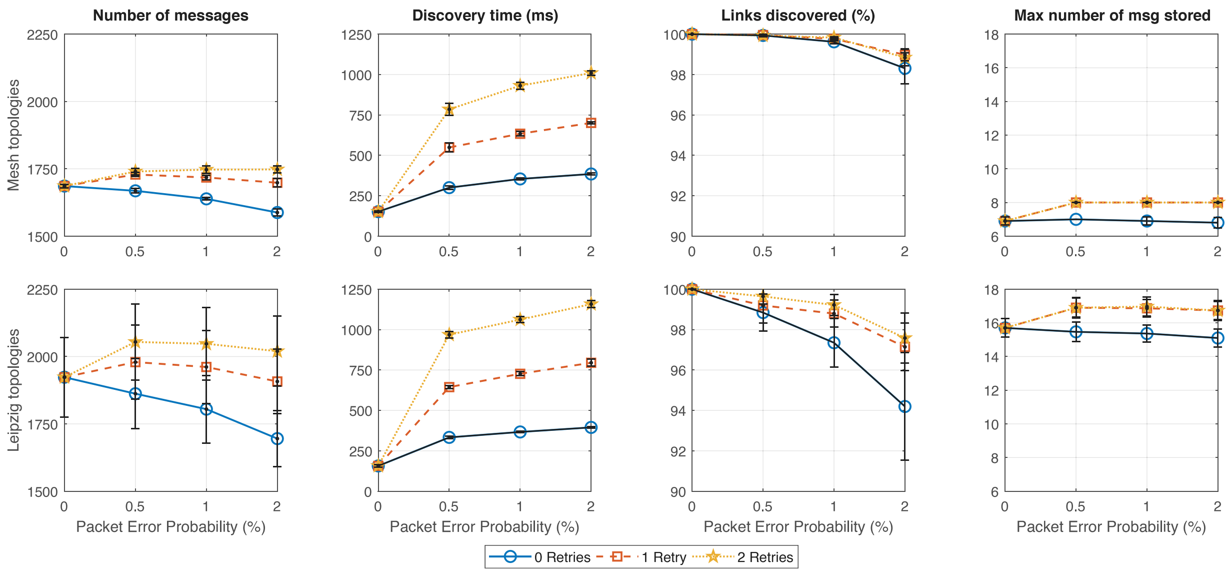
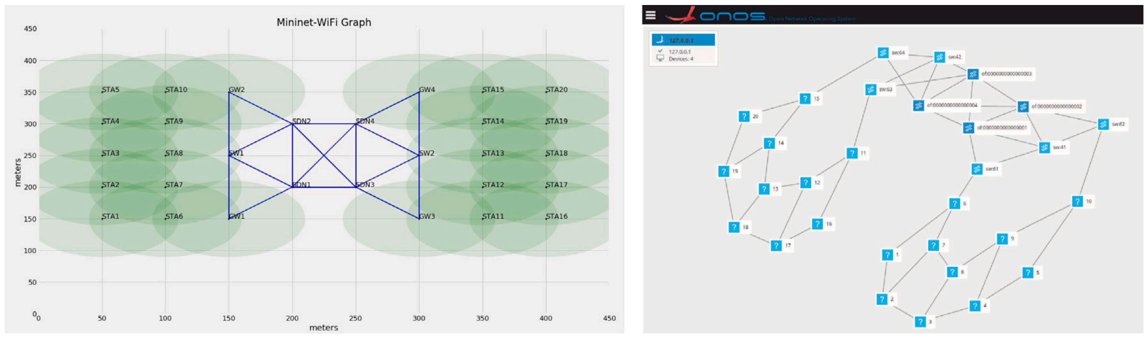


Fig. 16. Impact of packet error probability in Degree 4 topologies.



(a) Mininet (b) ONOS

Fig. 17. Full hybrid SDN network topology.

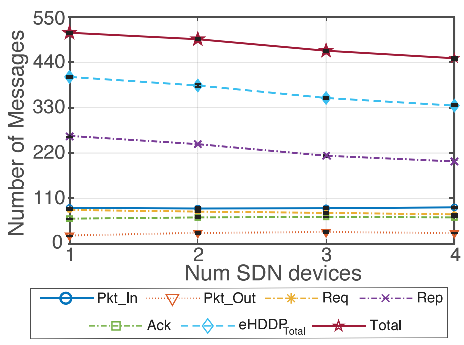
The results in terms of number of exchanged messages and discovery and processing times are summarised in Fig. 18. The figure shows the experiments with 1, 2, 3 or 4 SDN devices (devices in dark blue in Fig. 17) always placed at the core of the topology. The number of exchanged messages shown in Fig. 18(a) follows the same behaviour as in wireless topologies, decreasing linearly when the number of SDN devices increases, due to less hops are required to notify the information to the control plane. Regarding discovery and processing times, Fig. 18(b) shows lower values than the wireless scenarios but higher values than the wired scenarios with similar number of devices. This is the expected result since the exploration process time is the combination of the wired zone delay plus the wireless zone delay.

7. Conclusions

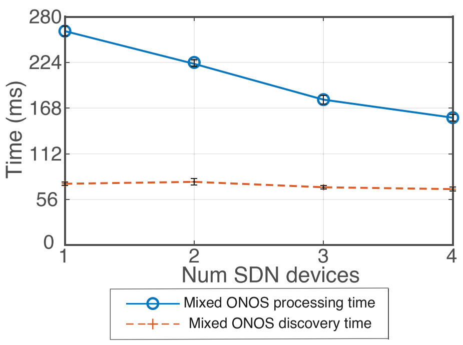
This paper presents eHDDP, a novel approach to gather the information from full-hybrid SDN topologies, which means that is not only able to discover hybrid SDN topologies with SDN and non-SDN devices, but also it is able to deal with wired and wireless links. This fact allows to integrate the whole information from the layer 2 in the SDN control plane, including devices with both wired and wireless interfaces. Hence, not only it enables the management of a complex core network, but it even allows to gather information from wireless networks connected to the wired infrastructure. This capability of integrating information from wired and wireless networks is the first step to combine the IoT world and WSNs with the SDN technology, opening up a huge market for new applications and services. Moreover, the information collected from the infrastructure layer allows to distinguish between bidirectional and unidirectional wireless links, a feature highly useful for optimal communications in wireless networks. In addition to eHDDP, this paper also presents an enhanced version of the Mininet-WiFi framework to properly support wireless Ad-Hoc networks with several devices that allowed to carry out a deep evaluation of eHDDP.

The obtained results demonstrate the scalability of the protocol since the required number of messages is proportional to the number of links in the network topology and always lower than state-of-the-art solutions such as OFDP in SDN scenarios. Moreover, the discovery time is in the range of hundreds of milliseconds opening the support for low mobility devices in SDN networks. Regarding the processing times at the controller, they are higher but can be reduced by increasing the computational resources accordingly.

As future work, we are planning to reuse the exploration process in eHDDP to not only discover any underlying topology but also for automatically establishing in-band SDN communication channels



(a) Number of messages



(b) Discovery and processing times

Fig. 18. Results in Full-hybrid topologies.

even on wireless SDN networks. This contribution would also help the deployment of IoT applications and services based on wireless SDN devices. To the best of our knowledge, there is not any proposal capable of establishing automatically in-band communication channels in wired and/or wireless hybrid SDN networks. Moreover, we also plan to study the security of eHDDP since the use of the Sequence Number and the ACK number may not be enough protection for all kind of attacks. Finally, we will continue with the enhancements of Mininet-WiFi and to study the optimal value of the eHDDP timer between retries depending on the wireless technology in use or how to include the information from the wireless channels to optimise its performance, e.g., packet losses or SNR.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] H. Farhady, H. Lee, A. Nakao, Software-defined networking: A survey, Comput. Netw. 81 (2015) 79–95, http://dx.doi.org/10.1016/j.comnet.2015.02.014, URL http://www.sciencedirect.com/science/article/pii/S1389128615000614.

[2] J. Alvarez-Horcajo, E. Rojas, I. Martinez-Yelmo, M. Savi, D. Lopez-Pajares, Hddp: Hybrid domain discovery protocol for heterogeneous devices in sdn, IEEE Commun. Lett. 24 (8) (2020) 1655–1659.

[3] F.F. Pakzad, et al., Efficient topology discovery in software defined networks, in: International Conference on Signal Processing and Communication Systems, 2014, pp. 1–8.

[4] I. S. 802.1AB, IEEE standard for local and metropolitan area networks - station and media access control connectivity discovery, 2009.

[5] D. Hasan, M. Othman, Efficient topology discovery in software defined networks: Revisited, Procedia Comput. Sci. 116 (2017) 539–547.

[6] A. Azzouni, et al., sOFTDP: Secure and efficient topology discovery protocol for SDN, 2017, CoRR 1705.04527. URL http://arxiv.org/abs/1705.04527.

[7] L. Ochoa-Aday, C. Cervelló-Pastor, A. Fernández-Fernández, Ch. A distributed algorithm for topology discovery in software-defined networks, in: A Distributed Algorithm for Topology Discovery in Software-Defined Networks, Springer International Publishing, 2016, pp. 363–367.

[8] S. Khan, et al., Topology discovery in software defined networks: Threats, taxonomy, and state-of-the-art, IEEE Commun. Surv. Tutor. 19 (1) (2017) 303–324.

[9] E. Rojas, et al., TEDP: An enhanced topology discovery service for software-defined networking, IEEE Commun. Lett. 22 (8) (2018) 1540–1543, http://dx.doi.org/10.1109/LCOMM.2018.2845372.

[10] L. Ochoa-Aday, et al., eTDP: Enhanced topology discovery protocol for software-defined networks, IEEE Access 7 (2019) 23471–23487, http://dx.doi.org/10.1109/ACCESS.2019.2899653.

[11] L. Ochoa-Aday, et al., Current Trends of Topology Discovery in OpenFlow-Based Software Defined Networks, Tech. Rep., Universitat Politécnica de Catalunya. Departament d'Enginyeria Telemática, 2015, URL https://upcommons.upc.edu/handle/2117/77672. (Accessed 14 September 2020).

[12] G. Tarnaras, et al., Efficient topology discovery algorithm for software-defined networks, IET Netw. 6 (2017) 157–161.

[13] L. Yang, et al., Forwarding and Control Element Separation (ForCES) Framework, Tech. Rep. 3746, Internet Engineering Task Force, 2004, URL http://www.ietf.org/rfc/rfc3746.txt.

[14] A. Mahmud, R. Rahmani, Exploitation of OpenFlow in wireless sensor networks, in: Proceedings of 2011 International Conference on Computer Science and Network Technology, Vol. 1, 2011, pp. 594–600.

[15] T. Luo, H. Tan, T.Q.S. Quek, Sensor OpenFlow: Enabling software-defined wireless sensor networks, IEEE Commun. Lett. 16 (11) (2012) 1896–1899.

[16] 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, pp. 1282–1287.

[17] S. Babu, et al., A novel framework for resource discovery and self-configuration in software defined wireless mesh networks, IEEE Trans. Netw. Serv. Manag. 17 (1) (2019) 132–146.

[18] B. Trevizan de Oliveira, L. Batista Gabriel, C. Borges Margi, Tinysdn: Enabling multiple controllers for software-defined wireless sensor networks, IEEE Lat. Am. Trans. 13 (11) (2015) 3690–3696.

[19] A.-C. Anadiotis, L. Galluccio, S. Milardo, G. Morabito, S. Palazzo, Sd-wise: A software-defined wireless sensor network, Comput. Netw. 159 (2019) 84–95, http://dx.doi.org/10.1016/j.comnet.2019.04.029, URL http://www.sciencedirect.com/science/article/pii/S1389128618312192.

[20] 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, pp. 1–6, http://dx.doi.org/10.1109/SECONW.2015.7328145.

[21] 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, pp. 1–8, http://dx.doi.org/10.1109/WFCS.2019.8757933.

[22] R.C.A. Alves, C.B. Margi, F.A. Kuipers, Know when to listen: SDN-based protocols for directed IoT networks, Comput. Commun. 150 (2020) 672–686, URL http://www.sciencedirect.com/science/article/pii/S0140366419303275.

[23] R.Y. Azhari, E. Firmansyah, A. Bejo, Simple protocol design of multi-hop network in lora, in: 2019 International Seminar on Research of Information Technology and Intelligent Systems, ISRITI, 2019, pp. 177–181, http://dx.doi.org/10.1109/ISRITI48646.2019.9034662.

[24] J. Dias, A. Grilo, Lorawan multi-hop uplink extension, Procedia Comput. Sci. 130 (2018) 424–431, http://dx.doi.org/10.1016/j.procs.2018.04.063, URL http://www.sciencedirect.com/science/article/pii/S1877050918304216. The 9th International Conference on Ambient Systems, Networks and Technologies (ANT 2018)/The 8th International Conference on Sustainable Energy Information Technology (SEIT-2018)/Affiliated Workshops.

[25] C. Gomez, J. Paradells, Wireless home automation networks: A survey of architectures and technologies, IEEE Commun. Mag. 48 (6) (2010) 92–101, http://dx.doi.org/10.1109/MCOM.2010.5473869.

[26] E. Rojas, et al., All-path bridging: Path exploration protocols for data center and campus networks, Comput. Netw. 79 (2015) 120–132, http://dx.doi.org/10.1016/j.comnet.2015.01.002.
[27] Eder Leão Fernandes, Elisa Rojas, Joaquin Alvarez-Horcajo, Zoltàn Lajos Kis, Davide Sanvito, Nicola Bonelli, Carmelo Cascone, Christian Esteve Rothenberg, The road to BOFUSS: The basic OpenFlow userspace software switch, J. Netw. Comput. Appl. 165 (2020) 102685.
[28] P. Berde, et al., ONOS: Towards an open, distributed SDN OS, in: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, 2014.
[29] eHDDP source code in GitHub, 2021, URL https://github.com/NETSERV-UAH/eHDDP/. (Accessed 21 February 2021).
[30] ns-3 simulator, 2020, https://www.nsnam.org/. (Accessed 14 September 2020).
[31] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, T. Voigt, Cross-level sensor network simulation with cooja, in: Proceedings. 2006 31st IEEE Conference on Local Computer Networks, 2006, pp. 641–648.
[32] A. Dunkels, B. Gronvall, T. Voigt, Contiki-a lightweight and flexible operating system for tiny networked sensors, in: 29th Annual IEEE International Conference on Local Computer Networks, IEEE, 2004, pp. 455–462.
[33] S. Duquennoy, et al., Contiki-NG: The OS for next generation IoT devices, 2019, URL https://github.com/contiki-ng/contiki-ng. (Accessed 16 September 2020).
[34] B. Lantz, B. Heller, N. McKeown, A network in a laptop: rapid prototyping for software-defined networks, in: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, 2010, pp. 1–6.
[35] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, et al., The design and implementation of open vswitch, in: 12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, 2015, pp. 117–130.
[36] R.R. Fontes, S. Afzal, S.H.B. Brito, M.A.S. Santos, C.E. Rothenberg, Mininet-WiFi: Emulating software-defined wireless networks, in: 2015 11th International Conference on Network and Service Management, CNSM, 2015, pp. 384–389.
[37] B.P. Crow, I. Widjaja, J.G. Kim, P.T. Sakai, IEEE 802.11 wireless local area networks, IEEE Commun. Mag. 35 (9) (1997) 116–126.
[38] A.A. Ahmed, A lightweight software defined network for resilient real-time internet of things, Int. J. Comput. Sci. Netw. Secur. 19 (8) (2019) 1.
[39] IEEE, IEEE standard for low-rate wireless networks, IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011), 2016, pp. 1–709.
[40] D.J. Dubois, Y. Bando, K. Watanabe, H. Holtzman, Lightweight self-organizing reconfiguration of opportunistic infrastructure-mode WiFi networks, in: 2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems, 2013, pp. 247–256.
[41] T. Høiland-Jørgensen, J. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, D. Miller, The eXpress data path: fast programmable packet processing in the operating system kernel, in: ACM CoNEXT '18, 2018, pp. 54–66, http://dx.doi.org/10.1145/3281411.3281443.
[42] GENI, OpenFlow discovery protocol (OFDP), 2010, URL http://groups.geni.net/geni/wiki/OpenFlowDiscoveryProtocol. (Accessed 14 September 2020).
[43] M.A. Brown, Traffic control HOWTO, 2006, URL https://tldp.org/HOWTO/Traffic-Control-HOWTO/. (Accessed 21 February 2021).
[44] GÉANT, GÉANT Topology Map August 2017, Tech. Rep., GÉANT, 2017, URL https://www.geant.org/Resources/PublishingImages/GEANT_topology_map_august2017.pdf. (Accessed 14 September 2020).
[45] A.L. Barabási, R. Albert, Emergence of scaling in random networks, Science 286 (5439) (1999) 509–512.
[46] B.M. Waxman, Routing of multipoint connections, IEEE J. Sel. Areas Commun. 6 (9) (1988) 1617–1622.
[47] B. Milic, M. Malek, Npart - node placement algorithm for realistic topologies in wireless multihop network simulation, in: Proceedings of the 2nd International Conference on Simulation Tools and Techniques, Simutools '09, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, BEL, 2009, pp. 1–10, http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5669.

Isaias Martinez-Yelmo received the Ph.D. degree in telematics from the Carlos III University of Madrid, Spain, in 2010. After working as a Postdoctoral Assistant with the Carlos III University of Madrid, he became and remains as a Teaching Assistant with the Automatics Department, Alcalá University, Spain. His research interests include peer-to-peer networks, content distribution networks, vehicular networks, NGN, and the Internet protocols. Nowadays, he is especially interested in advanced switching architectures, software-defined networks and P4. He has participated in various competitive research projects funded by the Madrid Regional Government (Medianet, Tigre5, TAPIR, IRIS), National projects (CIVTRAFF), and European projects (CONTENT, CARMEN, and so on). His research papers have been published in highimpact JCR indexed research journals, such as the IEEE Communications Magazine, Computer Communications, and Computer Networks, among others. He is currently Associate Editor in the Journal of Telecommunication Systems from Springer. In addition, he has been a Reviewer for high quality conferences (i.e., IEEE INFOCOM) and scientific journals (the IEEE Transactions on Vehicular Technology, Computer Communications, Computer Networks, the ACM Transactions on Multimedia Computing, and so on). He was a Technical Program Committee Member of IEEE ICON, from 2011 to 2013.

Joaquin Alvarez-Horcajo obtained his master's degree in Telecommunications Engineering from the University of Alcala in 2017 and his Ph.D. degree from the University of Alcala in 2020. He was a test engineer at Telefonica then he joined the University of Alcala where he was awarded a grant for university professor training (FPU) at the University of Alcala. The areas of research in which he works include Software Defined Networks (SDN), hybrid networks, and new generation protocols. He has participated in various competitive projects funded through the Community of Madrid plan (TAPIR-CM, IRIS, TIGRE5-CM, EsPECIE, and SIMPSONS).

Juan Antonio Carral received his Telecommunication Engineering degree from Technical University of Madrid in 1993 and his Ph.D. degree from the University of Alcala in 2013. He was a lecturer and research assistant at Technical University of Madrid from 1994 to 1998 then he joined the University of Alcala in 1999 as an assistant professor, he promoted to associate professor in 2000. His research activities cover the fields of SDN networks, advanced Ethernet switching, MAC protocols for radio access networks and performance simulation. He has been involved in many international and national granted research projects related with these topics.

Diego Lopez-Pajares (received the master's degree in telecommunications engineering, in 2016. He has been a Researcher with the NIS-NETSERV Research Group, University of Alcala, since 2015, focusing on topics related to delay-tolerant and SDN networks, specifically low-latency routing solutions and the multipath problem. These topics comprise the basis of his Ph.D. In addition, he actively participates in several research projects, such as TAPIR-CM, IRIS, TIGRE5-CM, EsPECIE, and SIMPSONS.