Rojas, E., Álvarez Horcajo, J., Martínez Yelmo, I., Carral, J.A. & Arco, J.M. 2018, "TEDP: An enhanced topology discovery service for Software-Defined Networking", IEEE Communications Letters, vol. 22, no. 8, pp. 1540-1543.

Available at http://dx.doi.org/10.1109/LCOMM.2018.2845372

TEDP: An enhanced topology discovery service for Software-Defined Networking

Elisa Rojas, Joaquin Alvarez-Horcajo, Isaias Martinez-Yelmo, Juan A. Carral and Jose M. Arco

***Abstract*—Currently, Software-Defined Networking (SDN) platforms leverage Link Layer Discovery Protocol (LLDP) to discover the underlying topology. However, LLDP is suboptimal in terms of message load. In this paper, we present the Tree Exploration Discovery Protocol (TEDP), proving that shortest paths can be built at the same time that the topology information is gathered, without extra messages compared to LLDP. We also analyze two alternative implementations for TEDP and give insights into some features that SDN platforms should ideally provide for an efficient topology discovery service.**

***Index Terms*—SDN, OpenFlow, topology discovery, LLDP.**

I. Introduction

NETWORK services are one of the main benefits of the thriving Software-Defined Networking (SDN) paradigm. These services are reusable pieces of network functionality that may be leveraged by any SDN application. Topology discovery could be considered one of the central SDN services, fuelled by the global perspective of the logically centralized SDN controllers.

Currently, the *de facto* protocol to implement the topology discovery service is the Link Layer Discovery Protocol (LLDP) [1]. Nevertheless, LLDP only explores SDN devices, hence the rest of the network (e.g. host or non-SDN devices) should be indirectly detected. This task depends on the particular strategy of the SDN platform, usually based on traffic snooping. Additionally, LLDP was designed for a distributed approach and, as such, it is suboptimal for SDN in terms of message overhead.

In this article, we present the Tree Exploration Discovery Protocol (TEDP), which provides an enhanced topology service without extra messages compared to LLDP. To this purpose, we first examine the related work in Section II. Secondly, we describe, implement and evaluate TEDP in Sections III, IV and V, respectively. Finally, we conclude the analysis in Section VI.

II. Related work

Discovery of network nodes is essential to provide the global vision required in SDN for network management. Therefore, some of the most popular SDN platforms (such as Ryu [2], OpenDaylight (ODL) [3] and Open Network Operating System (ONOS) [4] offer it as a service.

Elisa Rojas, Joaquin Alvarez-Horcajo, Isaias Martinez-Yelmo, Juan A. Carral and Jose M. Arco are with Departamento de Automatica, University of Alcala, 28805, Alcala de Henares. Corresponding e-mail: elisa.rojas@uah.es

This work was funded by a grant from Comunidad de Madrid through Project TIGRE5-CM (S2013/ICE-2919) and the "Formacion del Profesorado Universitario (FPU)" program from the University of Alcala.

Distributed approaches to discover forwarding devices are the standardized LLDP [1] or the proprietary Cisco Discovery Protocol (CDP). The main centralized –SDN-based– approach is OpenFlow Discovery Protocol (OFDP) [5], which leverages LLDP to perform topology discovery in OpenFlow-based networks. Alternatively, other SDN platforms such as ODL use the Broadcast Domain Discovery Protocol (BDDP) instead, a variation of LLDP with a broadcast destination MAC address –instead of multicast–. It allows discovering links in networks where traditional and SDN switches coexist.

OFDP is not efficient, as it periodically sends a number of packets directly proportional to the number of switches, which limits scalability [6]. One of the first optimizations of OFDP is OFDPv2 [7], which aims to decrease the SDN controller load. To this purpose, it reduces the number of `PACKET_OUT` messages sent from the SDN platform by modifying the LLDP frames and preinstalling some rules in the network devices. But this work was revisited in [8], proving that the number of messages was actually higher, though still better than OFDP.

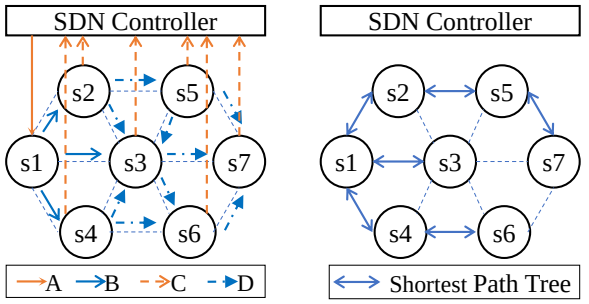
sOFTDP [6] relocates part of the discovery process in the switch, which memorizes topology information to asynchronously notify the controller based on specific events, instead of periodically, hence saving messages. The same asynchronous approach, but using ForCES [9] instead of OpenFlow, is also applied in [10]. Finally SD-TDP [11] is an optimization for OFDP in which network devices are assigned a hierarchy, so that only a few of them are in charge of obtaining the topological information, hence reducing the number of messages exchanged. Both sOFTDP and SD-TDP follow a hybrid –non-standard SDN– approach, a modification where part of the SDN intelligence is delegated to the switch.

Reference [12] presents a deep survey on topology discovery in SDN, describes potential threats, introduces a thematic taxonomy and provides a list of challenges and directions. TEDP is aligned with some of these directions, such as the autonomous discovery by switches. SHTD [13] embraces the same principle of delegating some services to the data plane, but focused on its self-healing properties.

Finally, in the case of wireless networks, alternatives to OFDP are described in [14], [15].

III. Tree Exploration Discovery Protocol

So far, all the presented proposals are based on neighbor discovery. Messages are exchanged point to point by switches, and this information is then shared with the logically centralized SDN controller. TEDP follows a totally different approach: instead of sending and receiving discovery messages



(a) Topology exploration in TEDP (simplified view).

(b) Shortest path tree towards switch A built with TEDP.

Fig. 1: TEDP behaviour example.

among neighbors, it simply sends a single *probe* frame that floods the network and explores its whole topology at once.

From this idea, two challenges arise:

1) **No-loop broadcasting:** What is the path that this frame should follow (if any) to reach all topology nodes? Can loops be prevented?
2) **Topology information conveyance:** Should this frame transport all the topology information encapsulated or just incremental updates?

To accomplish the first challenge, TEDP leverages the All-Path *locking* mechanism [16], which allows a frame to explore the network without loops. The only requirement is that network devices should be capable of processing the TEDP frame, as with LLDP. Regarding the second challenge, TEDP collects the topology information at each hop and sends it to the SDN controller, in charge of aggregating it. Thus, information gathering is performed exactly as in LLDP.

The difference between LLDP and TEDP, and an advantage at the same time, is that TEDP exploration frames traverse the network, hence fostering the opportunity to find optimal paths between pairs of nodes without additional cost.

A. Protocol operation

The operation of TEDP is summarized in Fig. 1a, divided into 4 steps. In step *A*, the SDN controller selects a target switch to start the TEDP discovery procedure from (switch `s1`). The TEDP frame (described afterwards) is sent as a `PACKET_OUT` to the switch and later on forwarded through all ports in step *B*. In step *C*, the incoming TEDP frame is sent to the SDN controller by all the network devices that receive it, i.e. all of its neighbors (`s2`, `s3` and `s4` in the figure).

Up to this point, the operation in TEDP is identical to LLDP. The difference appears in step *D* (dotted arrows), which defines that the TEDP frame must be forwarded to the rest of the network, until it reaches every device in it (repeating step *C* for every switch reached). As a consequence, the number of messages exchanged in the data plane is the same as in LLDP, but the process is triggered by a single switch in TEDP, thus requiring fewer control plane messages than LLDP.

Additionally, TEDP is able to gather latency-based paths in the procedure. To accomplish it, it saves the arrival port of the first copy of the frame. This port indicates the minimum latency path towards the source that originated the frame. For example, Fig. 1b depicts a possible shortest path tree created after applying the steps previously described. This tree represents the shortest path per each switch in the network towards switch `s1`, and viceversa. These paths are latency-based and supplement other types of shortest paths that could be calculated in the SDN controller after topology discovery, based on diverse metrics, such as hop count.

As in LLDP, the discovery process in TEDP is repeated periodically and it is topology-agnostic. If started at a different switch every time, TEDP eventually discovers minimum latency paths (tree) towards all switches. As only one tree is obtained in each iteration, to guarantee it is the fastest at a time, we should launch the process from that node on demand. Furthermore, it might not be necessary to run TEDP from all nodes in the network. For example, in networks where edge and core nodes are clearly differentiated, it would be much more advantageous to obtain updated trees for edge nodes, so the periodicity could be arranged depending on the node type. We can even ignore certain nodes and take advantage of the remaining cycles to provide better trees for the rest.

B. Discovery frame

Figure 2 compares both the LLDP (particularly for the Ryu controller implementation) and the TEDP frame. Both frames share the header, namely: a multicast destination address, the source address of the node that sends it and a specific `EtherType`. Regarding the payload, both require the `port` field to build the topology. The main difference is that, instead of the `Chassis ID` in LLDP (which usually represents the sender ID), TEDP conveys the MAC address to which the paths are being created (i.e. the node that originated the frame). The remaining payload fields are optional for TEDP, such as `TTL` and `End` fields. However, TEDP could employ the LLDP frame format if required.

IV. Implementation

As a proof of concept, we propose two different implementations of TEDP: a pure SDN service, where the implementation strictly resides at the controller (TEDP-S), and a hybrid one, where the service is developed in a shared mode between the SDN controller and the network switches (TEDP-H).

A. Service in the Ryu platform (TEDP-S)

Inspired by the practical implementation of LLDP in Ryu [2], we developed TEDP as a service in this platform. Similarly to LLDP, the code in TEDP is divided into three main phases, as depicted in Fig. 3. The first phase occurs at startup, where the controller obtains the relationship between each switch's ID and its MAC address. Then the controller installs the TEDP behaviour (step *A*) with a `FLOW_MOD` that produces a `PACKET_IN` sent to the controller (step *B*) and floods the TEDP frame (step *D*), as shown in Fig. 3b. Finally, Fig. 3c explains the procedure after receiving the `PACKET_IN` event (step *C*), which updates the topology and the exploration tree, and sends an additional `FLOW_MOD` to remove the flood rule, previously installed to prevent loops.

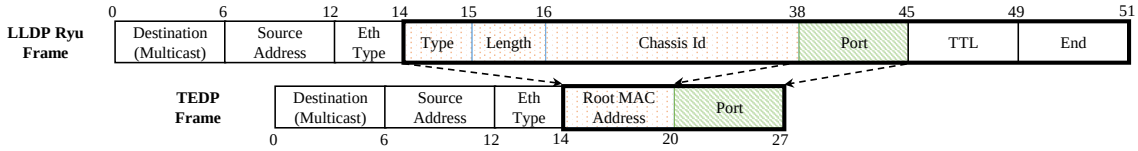


Fig. 2: TEDP frame in comparison with LLDP's

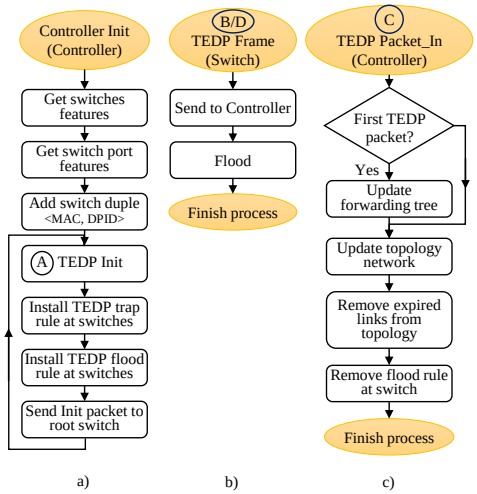


Fig. 3: Implementation logic of TEDP-S

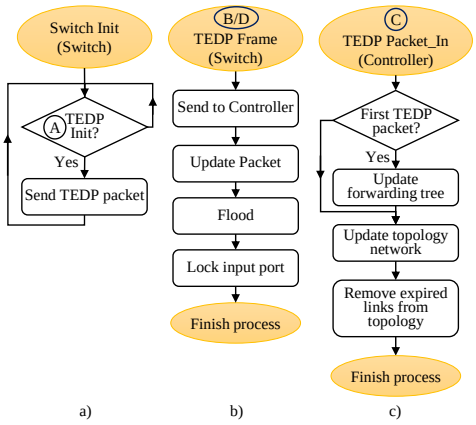


Fig. 4: Implementation logic of TEDP-H

B. Hybrid approach based on ofsoftswitch13 (TEDP-H)

Differently from the previous implementation, in the hybrid one (based on a modification of the *ofsoftswitch13* switch [17]), we delegate part of the controller intelligence to the network devices. Therefore, the SDN switches do not need an initial startup for the protocol, and can proceed to initiate the process and learn from it by themselves. The discovery is randomly started at one of the switches, which creates the TEDP frame and broadcasts it (step *A* in Fig. 4a). Every other switch in the network will notify its arrival to the controller (via a `PACKET_IN`), update the source address of the frame (with its own address), flood it and directly apply the *lock* (based on the `Root MAC` field) to avoid loops (step *B/D* in Fig. 4b). The `PACKET_IN` reports the captured information

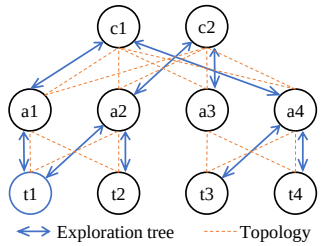


Fig. 5: TEDP exploration results in a Clos topology

to the controller, as shown in Fig. 4c (step *C*). The main difference with TEDP-S is that no `FLOW_MOD` is required this time from the controller in step *C*, as switches are autonomous enough to install their own forwarding rules.

V. Evaluation and Discussion

The evaluation of the two implementations of TEDP was performed in an infrastructure consisting of 2 hosts with Intel(R) Core(TM) i7 processors and 24 GB of RAM, running Ryu as SDN controller and Mininet. Each experiment was repeated 10 times, to compute mean and standard deviation.

A. Topology and shortest-path discovery

To evaluate the concepts and the implementation of TEDP, we first executed the protocol in diverse topologies to validate the correctness of both topology and shortest path discovery. These topologies include real world (see next section), data center, mesh and random networks. As an example, Fig. 5 illustrates one TEDP execution in a Clos (data center) topology composed of 10 switches (2 core, 4 aggregated and 4 top-of-rack switches). The topology was explored completely, and the shortest paths (as an exploration tree) were created correctly towards the originating switch, `t1` in Fig. 5.

B. Control and data message load

Secondly, we analyzed the expected (Table I) and experimental (Table II) number of messages generated both for LLDP and TEDP, as in [7]. To perform the comparison, we leveraged the GÉANT pan-European network topology (consisting of 44 nodes and 144 ports), as in SD-TDP [11].

TEDP and LLDP should require the same number of data plane messages, one per link in each direction, i.e. one per port (144 for GÉANT). Regarding the control plane, both protocols require the same number of `PACKET_IN` messages, one per port (144 as well), but a different number of `PACKET_OUT` and `FLOW_MOD` messages also depending on TEDP mode. TEDP-H does not need neither of them, as forwarding and

Packet #	Control Plane			Data Plane
	Packet_In	Packet_Out	Flow_Mod	
LLDP	144	144	44	144
TEDP-S	144	1	88	144
TEDP-H	144	0	0	144

TABLE I: Expected results of LLDP and TEDP

Packet #		Control Plane			Data Plane
		Packet_In	Packet_Out	Flow_Mod	
LLDP	$\overline{x}$	145,00	160.06	44,00	160.06
	σ	14,24	19,78	0,00	19,78
TEDP-S	$\overline{x}$	10,77K	1,00	10,82K	–
	σ	2,26K	0,00	2,26K	–
TEDP-S*	$\overline{x}$	170,63	1,00	227,44	170,63
	σ	2,31	0,00	1,71	2,31
TEDP-H	$\overline{x}$	146,00	0,00	0,00	146,13
	σ	0,73	0,00	0,00	0,81

TABLE II: Experimental comparison of LLDP and TEDP [18]

rule installation are performed by switches alone. TEDP-S requires one `PACKET_OUT` message to start the discovery and two `FLOW_MOD` messages per switch to install the forwarding rule and to remove it afterwards to prevent loops (88 for GÉANT). Finally, LLDP requires one `PACKET_OUT` per port, i.e. one per `PACKET_IN` received (144 for GÉANT), and one `FLOW_MOD` message per switch to install the LLDP behavior (44 for GÉANT). Table I summarizes these results.

Therefore, in both implementations and in comparison with LLDP, TEDP not only reduces the total number of messages exchanged, but it also adds the shortest path discovery functionality.

The experimental results of TEDP-H are extremely close to the expected ones (as shown in Table II). However, we found out a problem in the TEDP-S implementation: The *locking* mechanism (to stop flooding frames and prevent loops) is installed via a `FLOW_MOD` after receiving the first copy of the frame (as described in Fig. 3c), which is not fast enough to stop the frame copies arriving in the meantime. Thus, these later copies generate new `PACKET_IN` (and hence `FLOW_MOD`), flooded as well, creating a temporary explosion of frames that eventually ceases when the first `FLOW_MOD` arrives.

This behavior is due to the limitations of the OpenFlow protocol, which only allows the installation of the *lock* via the combination of a `PACKET_IN` plus a `FLOW_MOD`. We envision two modifications of OpenFlow that could solve this issue: (1) adding a counter to time out entries (e.g. removing the flooding action after it is used once), or (2) allowing the installation of table entries after a packet match (e.g. for installing the *lock* rule after the first matching frame). As a reference, we added a filtered version of TEDP-S in Table II as TEDP-S* (removing the extra `PACKET_IN` and `FLOW_MOD` messages generated due to this limitation).

VI. Conclusion

We have defined, implemented and evaluated TEDP. This protocol initiates the topology discovery at a single node, by flooding a probe frame to explore the network and collect its information, instead of polling each device and aggregating the replies afterwards, as in LLDP. The results are encouraging, not only the number of control messages are reduced, but the topology service is enhanced to provide latency-based paths.

Nevertheless, the change of approach proposed by TEDP has also disclosed some constraints in the current SDN implementations. For this reason, our conclusion is twofold. Firstly, network services should be specifically redesigned for SDN from scratch; it is our opportunity to think out of the box, instead of simply migrating old protocols. Secondly, the SDN architecture is still flourishing and should capture new ideas to reformulate its foundations, such as the OpenFlow protocol.

References

[1] I. S. 802.1AB, "IEEE Standard for Local and Metropolitan Area Networks - Station and Media Access Control Connectivity Discovery," 2009.

[2] Ryu. (2012) Ryu SDN framework. [Online]. Available: http://osrg.github.com/ryu/

[3] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a Model-Driven SDN Controller architecture," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, June 2014, pp. 1–6.

[4] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an Open, Distributed SDN OS," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, 2014.

[5] GENI, "OpenFlow Discovery Protocol (OFDP)," 2010. [Online]. Available: http://groups.geni.net/geni/wiki/OpenFlowDiscoveryProtocol

[6] A. Azzouni, R. Boutaba, T. M. T. Nguyen, and G. Pujolle, "sOFTDP: Secure and Efficient Topology Discovery Protocol for SDN," *CoRR*, vol. abs/1705.04527, 2017.

[7] F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska, "Efficient Topology Discovery in OpenFlow-based Software Defined Networks," *Comput. Commun.*, vol. 77, no. C, pp. 52–61, Mar. 2016.

[8] D. Hasan and M. Othman, "Efficient Topology Discovery in Software Defined Networks: Revisited," *Procedia Computer Science*, vol. 116, no. Supplement C, pp. 539 – 547, 2017, discovery and innovation of computer science technology in artificial intelligence era: The 2nd International Conference on Computer Science and Computational Intelligence (ICCSCI 2017).

[9] L. Yang, R. Dantu, T. Anderson, and R. Gopal, "Forwarding and Control Element Separation (ForCES) Framework," RFC 3746 (Informational), Tech. Rep. 3746, April 2004.

[10] G. Tarnaras, E. Haleplidis, and S. Denazis, "SDN and ForCES based optimal network topology discovery," in *Proceedings of the 1st IEEE Conference on Network Softwarization (NetSoft)*, April 2015, pp. 1–6.

[11] L. Ochoa-Aday, C. Cervelló-Pastor, and A. Fernández-Fernández, *A Distributed Algorithm for Topology Discovery in Software-Defined Networks*. Springer International Publishing, 2016, pp. 363–367.

[12] S. Khan, A. Gani, A. W. A. Wahab, M. Guizani, and M. K. Khan, "Topology Discovery in Software Defined Networks: Threats, Taxonomy, and State-of-the-Art," *IEEE Communications Surveys Tutorials*, vol. 19, no. 1, pp. 303–324, Firstquarter 2017.

[13] L. Ochoa-Aday, C. Cervelló-Pastor, and A. Fernández-Fernández, "Self-healing Topology Discovery Protocol for Software Defined Networks," *IEEE Communications Letters*, 2018.

[14] N. Abdolmaleki, M. Ahmadi, H. T. Malazi, and S. Milardo, "Fuzzy topology discovery protocol for SDN-based wireless sensor networks," *Simulation Modelling Practice and Theory*, vol. 79, no. Supplement C, pp. 54 – 68, 2017.

[15] L. Chen, S. Abdellatif, P. Berthou, B. Nougnanke, and T. Gayraud, "A Generic And Configurable Topology Discovery Service For Software Defined Wireless Multi-Hop Network," in *15th ACM International Symposium on Mobility Management and Wireless Access*, Miami, United States, Nov. 2017, p. 4p.

[16] E. Rojas, G. Ibáñez, J. M. Gimenez-Guzman, J. A. Carral, A. Garcia-Martinez, I. Martinez-Yelmo, and J. M. Arco, "All-Path bridging: Path exploration protocols for data center and campus networks," *Computer Networks*, vol. 79, pp. 120 – 132, 2015.

[17] "CPqD/ofsoftswitch13." [Online]. Available: https://github.com/CPqD/ofsoftswitch13

[18] E. Rojas, J. Alvarez-Horcajo, I. Martinez-Yelmo, J. A. Carral, and J. M. Arco, "IEEE Data Port - Dataset of TEDP (experiments and results)," 2018. [Online]. Available: http://dx.doi.org/10.21227/H2XT00