Discovering the Network Topology: An Efficient Approach for SDN

Leonardo Ochoa-Aday[a], Cristina Cervelló-Pastor[b], and Adriana Fernández-Fernández[a]

[a]Sudent Member, IEEE. Department of Network Engineering, Universitat Politècnica de Catalunya (UPC), Esteve Terradas, 7, 08860, Castelldefels, Spain

[b]Member, IEEE. Department of Network Engineering, Universitat Politècnica de Catalunya (UPC), Esteve Terradas, 7, 08860, Castelldefels, Spain {leonardo.ochoa, cristina, adriana.fernandez}@entel.upc.edu

KEYWORD

SDN; Agents; Topology discovery; Distributed protocol

ABSTRACT

Network topology is a physical description of the overall resources in the network. Collecting this information using efficient mechanisms becomes a critical task for important network functions such as routing, network management, quality of service (QoS), among many others. Recent technologies like Software-Defined Networks (SDN) have emerged as promising approaches for managing the next generation networks. In order to ensure a proficient topology discovery service in SDN, we propose a simple agents-based mechanism. This mechanism improves the overall efficiency of the topology discovery process. In this paper, an algorithm for a novel Topology Discovery Protocol (SD-TDP) is described. This protocol will be implemented in each switch through a software agent. Thus, this approach will provide a distributed solution to solve the problem of network topology discovery in a more simple and efficient way.

1. Introduction

The huge demand of Internet services such as Big Data, Cloud services, Internet of Things (IoT), among others, are the main drivers for the need of a new paradigm in networking. This significant amount of traffic has been increasing on the same level as the number of network user and communication devices, such as terminal computers, smartphones and sensors (Jammal et al., 2014). Answering these exponential demands dynamically and efficiently, is one of the biggest challenges for the network management in current networks (Jammal et al., 2014). A considerable effort to address these issues is the new networking paradigm called Software-Defined Networks (SDN) (Nunes et al., 2014). This emerging architecture promises to simplify the network management while allowing to innovate and develop on the network in simpler way (Nunes et al., 2014).

In SDN the global view of the network is provided by the network controller through the topology discovery service (Kreutz et al., 2015). Discovering the network topology is critical for several controller services such as routing, network management, resource allocation and configuration, quality of service (QoS), diagnosis and fault recovery. In that direction, minimizing the time for obtaining the network topology is fundamental to make timely and decisive responses according to real-time events. Therefore, discovering the up-to-date topology of the network using efficient mechanisms constitutes a compulsory task for every network operator and it also becomes one of the most significant design metric for large-scale SDN.

Maintaining a comprehensive view of large networks generates a considerable amount of state information from the physical plane (Aslan and Matrawy, 2016). Furthermore, substantial volume of control traffic would represent a considerable pressure for the central controller and, as a consequence, scalability issues might appear (Levin et al., 2012). Schemes where each forwarding node has to send the topology information periodically could overload the controller performance (Aslan and Matrawy, 2016). To solve this problem our

research aims to design a simple and efficient mechanism for discovering the link layer in SDN. The proposed solution uses a distributed algorithm and switches supporting software agents for minimizing both the time and number of packets required to obtain the network graph. This approach is intended to get this graph in large-scale networks decreasing the overload in the controller performance as well as to enforce the current topology discovery service in SDN.

Discovering the physical topology requires to divide the big discovery problem into smaller processes. In that way and without incurring in scalability issues, we could obtain the network graph as quick as possible. Our research hypothesis is that an efficient and simple mechanism for topology discovery in large scale SDN could be achieved through careful design of an algorithm which divides the whole process in phases and distributes hierarchically the discovery functions between the network nodes.

In this study, we propose a distributed algorithm that is based on a simple agents-based mechanism to improve the efficiency of the topology discovery process. This algorithm will be used in the design of a novel topology discovery protocol, called **S**oftware **D**efined Network - **T**opology **D**iscovery **P**rotocol (SD-TDP). This protocol will be implemented in each switch through a software agent. Thus, this approach will provide a distributed solution considering that the topology discovery process is performed by nodes that support the network protocol.

The remainder of this work is organized as follows. In Section 2, we show a brief review about the state-of-the-art and related works. In Section 3, we describe the technical proposal. In Section 4, we evaluate the preliminary results achieved by this approach so far. Finally, Section 5 the main conclusions and future studies of this work are outlined.

2. Background

Efficient procedures for topology discovery in SDN have been already researched in (Pakzad et al., 2014; Pakzad et al., 2016). The topology discovery mechanisms proposed by the authors have focused on improving the current OpenFlow-based SDN mechanism.

The authors of (Pakzad et al., 2014; Pakzad et al., 2016) evaluate the efficiency of the current OFDP mechanism implemented by major SDN controllers. Moreover, they have proposed simple and practical modifications to reduce the controller overhead during the topology discovery procedure. In (Pakzad et al., 2014), after implementing the improved approach in POX controller and Mininet emulator, results show a reduction in the controller overload up to 45%. On the other hand, testing the proposed modification in a specific topology in OFELIA SDN testbed (Pakzad et al., 2016), show a reduction in the controller overload up to 40%, while the physical topology is showed in the same fashion.

These works propose improving the efficiency of the OpenFlow-based mechanism based on the reduction of packet_out messages sent from the controller to OpenFlow switches. Furthermore, the authors implement an OFDPv2 based on the ability of OpenFlow switches to rewrite packet headers. As a result, the number of packet_out messages sent by the controller was reduced to only one message per OpenFlow switch.

We have also analyzed some research contributions which discover the network topology in SDN using non-OpenFlow mechanisms.

In (Tarnaras et al., 2015), the authors proposed an automatic topology discovery algorithm taking into account a better usage of the LLDP protocol. This protocol is used locally in the data plane of switches for updating periodically their local neighbors table. Afterwards, they obtained the topology map, using the IETF ForCES framework for extracting the LLDP data directly from the network devices (i.e. LLDP local system MIB). This procedure automatically reports to the controller any change in the physical topology as a triggered event. After implementing the proposed algorithm, the simulation results showed that the average time to discover a new switch (i.e. 12 ms) during the topology discovery process is 90% less than OpenFlow-based solution (i.e. 100 ms).

In (Jiménez et al., 2015), the authors propose SDN Resource Discovery Protocol (SDN-RDP) as an alternative

to distribute the management of the network state among several SDN controllers. Each controller discovers a portion of the network topology in order to ensure the distribution of the node management and also simplify the protocol resolution. The described mechanism is asynchronous without a global initialization process and neither need previous knowledge of the network. Based on simulation results, the proposed protocol achieves an efficient reduction of the controller overload. On the other hand, fault recovery in the data layer could be achieved within the 50 ms required in transport networks.

The papers mentioned above (Pakzad et al., 2014; Pakzad et al., 2016; Tarnaras et al., 2015) need each SDN switch has preconfigured a network identification (i.e. IP address). Otherwise, the described mechanisms will not discover the network topology. Furthermore, OpenFlow controllers also require a previous knowledge of some parameters of active SDN switches in the network such as number of ports, MAC address, among others. This information is requested by the controller after the establishment of the initial OpenFlow connection with each forwarding device. Based on this, the OpenFlow controller implements a topology discovery mechanism that uses the frame format defined by the Link Layer Discovery Protocol (LLDP). With the exception of the frame format, this topology discovery mechanism has not much in common with LLDP. In general, there is a lack of topology discovery mechanisms in SDN based on layer 2 techniques.

Contrary to these related works, we propose a topology discovery mechanism that enables discovering the network nodes before the initial connection between the controller and the SDN switches. This idea is also considered in (Jiménez et al., 2015), but we have also proposed that selected nodes aggregate the topology information and send it to the controller in order to improve the overall efficiency of the topology discovery process. In this way, our mechanism discovers the network topology without the need of previous configurations nor previous controller knowledge of the network, which are the main limitations found in the literature. This procedure enables SDN controllers to discover the network topology using layer 2 techniques and be connected in already deployed networks, as an approach for fast installation.

3. Technical Proposal

Network topology discovery is a representation of how devices in network or sub-networks are physically connected (Alhanani and Abouchabaka, 2014). In that direction, we propose a mechanism to automatically discover the network topology without configuring the IP layer in the network. This novel approach is designed to discover the network topology based only on layer 2 techniques.

To improve the overall efficient of the topology discovery service, we propose that selected nodes aggregate the topology information and send it to the controller. In this way, we distribute the discovery functions between the network nodes. Each forwarding node has to support the proposed algorithm through an agent. Previous selection of the nodes that respond back to the controller, will be more efficient than approaches where every node in the network has to send topology data toward the controller. The subset of selected nodes is such that, each node in the network is either a part of the subset or is associated to a node in the subset.

3.1 Mechanism Description

The whole mechanism is divided into two phases. The first phase is initialized by the controller sending a multicast message, called *TDP-Request*, through all the active interfaces. This stage lasts during the propagation of the message through the network. The node identifier used in the packets will be the MAC address. The agents within the switches change from the initial state (*Standby*) to the corresponding one according to the state machine of Figure 1. As a result, the distributed algorithm defines a hierarchical delay-constrained shortest path tree rooted at the controller and, simultaneously, sets which switches will send the topology data to the controller. In addition, each node will know the MAC address of its controller.

In the second phase, selected switches (*Father nodes, FN*) aggregate topology data from their neighbors (*Active nodes, AN*) and asynchronously send it to the controller through the hierarchical tree. Each *FN* sends the topology data when it has the information from all the *AN* associated to it. Moreover, each *AN* has only one *FN*, at which it will send its topology information.

Each node in the network, after receiving a *TDP-Request* message, has to reply this request to the sender node and then forwards the *TDP-Request* through its other interfaces. Based on this mechanism, each node can determine the delay to its neighbours. Furthermore, each *AN* has to send a *TDP-ACK* message to indicate its association with its *FN*.

3.2 SD-TDP State Machine

To better understand the behavior of the switches with SD-TDP based agent, we present the operation of the software agents within the forwarding nodes as a state transition diagram. This state diagram shows all possible switch states drawn as circles. The arcs represent transitions from one state to another and are labeled with the condition that will change the state.

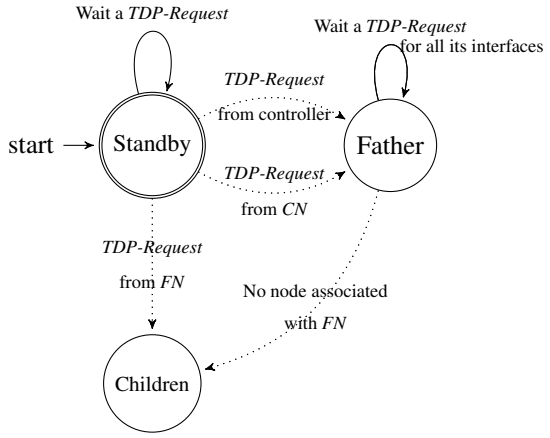

Figure 1: State machine of a switch with SD-TDP based agent

Initially, each switch in the network is in the *Standby* state, waiting for a *TDP-Request* message from another node (i.e. switch or controller). Once a *TDP-Request* is received, the switch with *Standby* state examines the packet and changes to either the *Father* state or the *Children* state, according to the sender state value found in the packet. The possible sender state values (i.e. Controller, *Father* and *Children*) are shown as transition arc labels from the *Standby* state.

Therefore, the resulting states only depend on the sender state value represented in the *TDP-Request* message. For instance, if a *TDP-Request* message from a sender with *Father* state is received, the switch changes from the initial state (i.e. *Standby*) to *Children* state. On the other hand, if the switch receives a *TDP-Request* message from the controller or a sender with *Children* state, the switch changes to the *Father* state. It is important to highlight that, the switches will change its state after receiving the first *TDP-Request* message, only if they have the *Standby* state.

As a key issue for improving the overall efficiency of SD-TDP, we have defined that nodes with *Father* state without any node associated to it, will automatically change to *Children* state. To achieve this, when a switch reaches the *Father* state, it waits for a *TDP-ACK* message by at least one of its active interfaces. If after receiving

all the corresponding messages from its neighbor nodes, no *TDP-ACK* message was received, the switch changes to the *Children* state. Otherwise, the switch remains in the *Father* state. In this way, the controller overhead produced by the sending of topology data is decreased.

4. Evaluation

In this section, we have developed a heuristic algorithm to evaluate the feasibility of the proposed mechanism. We have implemented this algorithm using the programming language Python and the NetworkX library (NetworkX, 2016) for graph modelling. Moreover, we carried out several experimental simulations with real-world topologies from the Internet Topology Zoo (Knight et al., 2011).

Algorithm 1 Finding an optimal-delay spanning tree

Require: G network topology, C controller position in G
Ensure: P_T optimal-delay paths, S nodes that send topology data
1: $Q \leftarrow Qinit()$ ▷ create the priority queue
2: $P_T, S \leftarrow None$
3: **for** $n \in N$ **do**
4: $Qinsert(n)$
5: **if** $n = C$ **then**
6: $S \leftarrow n$
7: $n.distto \leftarrow 0$
8: $n.predec \leftarrow$ set predecessor C
9: **end if**
10: $n.distto \leftarrow \infty$ ▷ set distance to the controller
11: $n.predec \leftarrow None$
12: **end for**
13: **while** Q is not empty **do**
14: $n = Qdelmin(Q)$
15: **for** edge (n, v) such that v is in Q **do**
16: **if** $v.distto > n.distto + d_{n,v}$ **then**
17: $v.distto = n.distto + d_{n,v}$
18: **if** $v.predec$ not in S **then**
19: $S \leftarrow v$
20: **end if**
21: $Qdecdistto(v, v.distto, Q)$
22: $v.predec \leftarrow$ set predecessor n
23: **end if**
24: **end for**
25: **end while**

4.1 Algorithm Description

To evaluate the results achieved by the proposed mechanism, we have developed an extension of Dijkstra algorithm for supporting *Father* switch selection, showed in Algorithm 1. Furthermore, we have analyzed the complexity of our implementation in this modified algorithm.

This heuristic algorithm considers a physical network modeled by a directed connected graph $G = (V, E)$, with V a set of N nodes and E a set of edges. Each edge $(i, j) \in E$ has associated a nonnegative delay $d(i, j)$. Let denote C as the controller located at one specific node in the graph. The goal is to find the subset of paths P_T, which forms an optimal delay-constrained tree rooted at the controller. Furthermore, the subset of selected nodes $S \subset N$ to respond back toward the controller (i.e. set of *Father* nodes) with topology information is minimized.

The algorithm is implemented using a priority queue (fibonacci heap), in order to accomplish a good performance on deployments with limited computation time. This procedure begins initializing the priority queue and after that, we set initial parameters as distance to controller ($distto$) and predecessor ($predec$) to each network node, and also store the nodes in the heap. Afterwards, we select the SDN controller as the starting point to ensure that the optimal-delay spanning tree have as a root the controller. Subsequently, we get the node with minimum $distto$ and delete it from the priority queue through the function in line 14. For each adjacent edge (n, v), if the neighbour v is in the heap and its $distto$ is higher than $(n.distto + d_{n,v})$, being $d_{n,v}$ the link-delay among the nodes, we update the priority queue with the computed distance to the controller of the neighbour v and, simultaneously, set n as predecessor of v. This procedure adds an edge to the desired tree in every iteration. The functions used in lines 1, 4, 14, 21 are classical implementations of the fibonacci heap and their time complexity is well described in the literature (Brodal et al., 2012).

4.1.1 Complexity

The time complexity of this heuristic algorithm is affected by the data structure used in the implementation. Therefore, if we use only an adjacent list for this algorithm, we can define the asymptotic complexity as $\mathcal{O}(N^2)$, being N the number of nodes in the network. However, we have implemented this algorithm using a priority queue (fibonacci heap) and thus, its complexity has been improved to $\mathcal{O}(E + N \cdot \log N)$, being E the number of links among the switches. This result agrees with the lowest bound achieved by Dijkstra heuristic algorithm, using the same data structure.

We also analyze the overall complexity of our mechanism in terms of number of messages sent. Considering N the number of nodes in the network, and V_n the number of neighbours of a node $n \in N$, we can express the number of messages sent as $M = N \cdot \sum_{n \in N} V_n + N$. Due to V_n reaches his maximum value in fully connected networks, the asymptotic overall complexity of the messages sent in the network can be considered of $\mathcal{O}(N^2)$.

4.2 Preliminary Results

Motivated by the results, we further evaluate the proposed algorithm using real topologies contained in the Internet Topology Zoo dataset, assuming each node in the topology as an SDN switch. We conducted all the simulations on a computer with 3.30 GHz Intel Core i7 and 16 GB RAM.

In Figure 2, some metrics of running the SD-TDP protocol on GÉANT (40 nodes, 61 links) topology are shown. This figure reveals the time for discovering all nodes and the number of *FN* that send topology data. Each point in Figure 2 represents a simulation taking one node of the topology as network controller. SD-TDP shows better performance with Germany as controller, it is also worth nothing that these results coincide with the controller placement obtained using the minimum K-center algorithm (Heller et al., 2012).

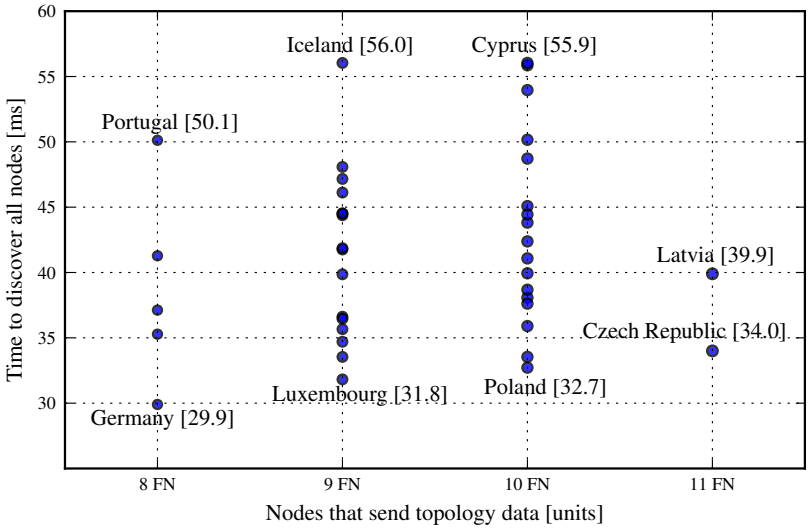

Figure 2: Parameters of running SD-TDP on GÉANT topology

Running SD-TDP on GÉANT has shown that the distributed protocol converges and discovers all nodes in 29.9 ms, with Germany as controller. Moreover, the algorithm reveals that only 8 *FN* of 40 total nodes had to send topology data to the controller, needing in total 162 packets to achieve a global view of the network.

In addition, results of Figure 2 show the minimum convergence time of the entire process. These times are equal to the Round Trip Time (RTT) from the controller to the farthest node, achieved through the hierarchical tree formed in the first phase.

5. Conclusions

In this paper, we propose a distributed mechanism for discovering layer 2 topologies in large SDN using an agents-based mechanism. This mechanism enables automatic discovering of the network without the need of previous IP configurations nor controller knowledge of the network, as has been considered in the literature. Through experimental simulations with real world topologies we demonstrate that SD-TDP provides a suitable approach for discovering the network topology. The results after running the algorithm on GÉANT show that the convergence time of our proposal is upper bounded by $\mathcal{O}(E + N \cdot \log N)$, with a simple and efficient scheme.

Our future efforts will be directed to generalize the proposed protocol for supporting multi-domains SDN and decreasing dynamically the control plane overhead associated with network state and topology information. In addition, we pretend to incorporate fault-tolerance mechanisms as an approach to ensure robustness of the control plane against changes in the network.

Acknowledgements

This work has been supported by the Ministerio de Economía y Competitividad of the Spanish Government under project TEC2013-47960-C4-1-P.

6. References

Alhanani, R. A. and Abouchabaka, J., 2014. An Overview of Different Techniques and Algorithms for Network Topology Discovery. In *Second World Conference on Complex Systems (WCCS), 2014*, pages 530–535.

Aslan, M. and Matrawy, A., 2016. On the Impact of Network State Collection on the Performance of SDN Applications. *IEEE Communications Letters*, 20(1):5–8. ISSN 1089-7798.

Brodal, G. S., Lagogiannis, G., and Tarjan, R. E., 2012. Strict Fibonacci Heaps. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 1177–1184. New York, USA. ISBN 978-1-4503-1245-5.

Heller, B., Sherwood, R., and McKeown, N., 2012. The Controller Placement Problem. *SIGCOMM Comput. Commun. Rev.*, 42(4):473–478. ISSN 0146-4833.

Jammal, M., Singh, T., Shami, A., Asal, R., and Li, Y., 2014. Software Defined Networking: State of the Art and Research Challenges. *Computer Networks*, 72:74–98.

Jiménez, Y., Cervelló-Pastor, C., and García, A., 2015. Dynamic Resource Discovery Protocol for Software Defined Networks. *IEEE Communications Letters*, 19(5):743–746. ISSN 1089-7798.

Knight, S., Nguyen, H. X., Falkner, N., Bowden, R., and Roughan, M., 2011. The Internet Topology Zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775. ISSN 0733-8716.

Kreutz, D., Ramos, F. M. V., Veríssimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S., 2015. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 103(1):14–76. ISSN 0018-9219.

Levin, D., Wundsam, A., Heller, B., Handigol, N., and Feldmann, A., 2012. Logically Centralized?: State Distribution Trade-offs in Software Defined Networks. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 1–6. New York, USA. ISBN 978-1-4503-1477-0.

NetworkX, 2016. Available in: http://networkx.readthedocs.io/en/networkx-1.11/. Accessed on 03/03/2016.

Nunes, B. A. A., Mendonca, M., Nguyen, X. N., Obraczka, K., and Turletti, T., 2014. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. *IEEE Communications Surveys Tutorials*, 16(3):1617–1634. ISSN 1553-877X.

Pakzad, F., Portmann, M., Tan, W. L., and Indulska, J., 2014. Efficient Topology Discovery in Software Defined Networks. In *8th International Conference on Signal Processing and Communication Systems (ICSPCS), 2014*, pages 1–8.

Pakzad, F., Portmann, M., Tan, W. L., and Indulska, J., 2016. Efficient Topology Discovery in OpenFlow-based Software Defined Networks. *Comput. Commun.*, 77(C):52–61. ISSN 0140-3664.

Tarnaras, G., Haleplidis, E., and Denazis, S., 2015. SDN and ForCES Based Optimal Network Topology Discovery. In *1st IEEE Conference on Network Softwarization (NetSoft), 2015*, pages 1–6.