International Journal of Science and Research (IJSR)
ISSN: 2319-7064
ResearchGate Impact Factor (2018): 0.28 | SJIF (2018): 7.426

# Enhanced Topology Discovery Protocol for SDN

**Ajeesh S[1], Betty Mary Jacob [2], Nisha Mohan P M [3]**

[1, 2, 3]Mount Zion College of Engineering, Department of Computer Science and Engineering, Pathanamthitta, Kerala, India

**Abstract:** ***In software-defined networks (SDNs), the controller collects the topology information from the data plane and maintains an abstract view of the entire network, which is crucial for the proper functioning of applications and network services. However, there is still the need for an enhanced protocol for automatic discovery and mechanisms of auto configuration of network elements according to new policies and business requirements. To overcome this challenge, this paper presents a novel protocol that, unlike existing approaches, enables a distributed layer-2 discovery without the need for previous network configurations or controller knowledge of the network. By using this mechanism, the SDN controller can discover the network view without incurring scalability issues, while taking advantage of the shortest control paths toward each switch.***

**Keywords:** Network management, protocols, software-defined networks, topology discovery

## 1. Introduction

To be able to address these high demands from users, network operators will require emerging solutions to effectively manage their network resources in a dynamic and flexible manner. In addition, in order to deploy high-level policies in traditional networks, operators need to configure each element of the network. This often occurs via specific, low-level commands from manufacturers because the plane that determines how to manage traffic (the control plane) and the plane that forwards traffic in accordance with the decisions of the control plane (the forwarding plane) are vertically integrated into a single network device.

The term ‘‘programmable networks’’ is usually employed to describe the desired future of networking. In essence, a network is said to be programmable if the behavior of its network devices and its traffic control are managed by software that operates independently from the network’s physical infrastructure. Moving from closed, proprietary-based computer hardware to software-oriented (and thus programmable) networks provides the opportunity for networking innovation, making it possible and more straightforward to evolve network capabilities and deploy new services.

## 2. Topology Discovery Service in SDN

In general, topology discovery is highly important in several computer network areas such as routing, resource allocation and configuration, Quality of Service (QoS), network management, diagnosis and fault recovery, among others. For this reason, discovering the current topology of a network is a compulsory task for every network operator. Moreover, collecting this real-time information efficiently and automatically is critical for significant networking problems such as enhancing network connectivity and resolving network congestion. In order to improve the performance of these network services, preserving an accurate view of the network topology at all times is also an important task.

In order to discover the topology under OFDP, switches require two major previous configurations. Firstly, every switch has initially programmed the IP address and TCP port of its controller to establish a connection as soon as the device is turned on. Secondly, switches have preinstalled flow rules to route directly to the controller via a packet-in message, any LLDP packet received from another switch.

Using the IP address and TCP port programmed in advance, the switch searches for its controller in the network and attempts to establish a secure and encrypted connection through a Transport Layer Security (TLS) session. The controller as part of this initial handshake sends a feature request message to the switch, which responds with a feature reply message. With this message the switch informs the controller about relevant parameters for the network discovery such as the switch ID, a list of active ports with their corresponding MAC addresses, among others.

In essence, under OpenFlow, switches are discovered and added to the network view in the initial handshaking process. Consequently, with OFDP the topology discovery is reduced to discover the inter-connected links between the switches. Moreover, sending messages periodically from the controller to each OpenFlow switch increases the network traffic and latency between the control plane and the forwarding plane, and can also lead to network limitations and outages

## 3. Enhanced Topology Discovery Protocol

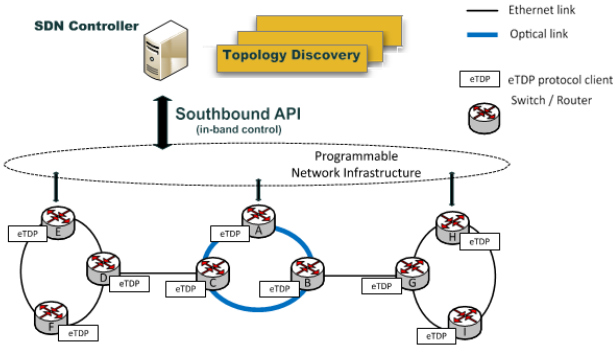
Although SDN provides a flexible architecture by centralizing network intelligence, the controller intervention in those control tasks that only require local switch knowledge is not always ideal. The execution of such tasks (e.g. neighbor discovery) can be delegated to the forwarding devices, which can gather the corresponding information and send it to the controller. In this way, the controller remains responsible for performing those tasks that require a global network view and centralized control.

### 3.1 Programmable Network Infrastructure

The proposed solution can be implemented in a network system following the recent design proposed by the SDN paradigm. The overall system architecture for the proposed solution is shown in Fig.. This network system embraces

network control decoupled from forwarding devices and leverages SDN controllers to provide an abstract view of the entire network.

This proposal can be deployed in a network domain with multiple SDN controllers through the use of a software agent (e.g. eTDP client) running in each network device. As a result, based on the topology information sent by switches each SDN controller discovers and maintains an accurate network view in the topology database. The stored information is critical for the proper operation of other controller services and supported network applications (e.g. traffic engineering,




### 3.2 Forwarding Network Device

The network devices may be any hardware-based (i.e. switch or router) or software-based (logical or virtual- ized) device configured to perform data forwarding functions according to the routes specified by the SDN controller. Fig. 2 presents a schematic diagram of a network device.

The Topology Discovery Protocol Agent is the compo- nent responsible for performing the proposed eTDP at each node, which can be implemented using an agent-oriented approach. These agents perform a local partial function of the entire discovery process while interacting autonomously. This capability of distributed operation allows the global topology discovery task to evolve in a scalable and effective way, without overburdening the SDN controller. Topology information retrieved by each node during the eTDP oper- ation is temporarily stored in the device memory and periodi- cally sent to the controller of the corresponding SDN domain.

### 3.3 Control Frames Description

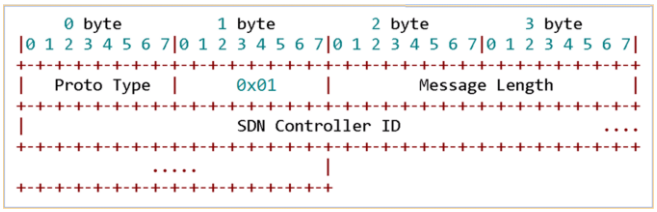
The eTDP communications are carried out using a stan- dard network frame format for all data related to the proto- col. This feature allows us to develop future extensions of the protocol while maintaining compatibility with previous versions. Moreover, the packets are encapsulated with their corresponding headers (i.e. MAC or IP) for transmissions over the network.

**1) Message Header**

These messages share a common header format, which allows a node to be able to accept or relay (if applicable) mes- sages of different types. This feature supports fine-grained message forwarding using the powerful ‘‘match action’’ abstraction of SDN. The definitions of each field included in the message headers are further described below:

- *Proto Type*: Protocol type (8 bits). This field uses a specific hexadecimal number to denote the protocol type so that any switch that supports this protocol can easily identify eTDP messages in the network control frame.
- *PDU Type*: Packet data unit type (8 bits). This field contains a value that specifies the type of message in the payload. For example, type 0x01 denotes a topoRe- quest frame, type 0x02 indicates an echoReply frame and type 0x03 corresponds to a topoReply frame.
- *Message Length*: Message size (16 bits). This field indicates the message end in the byte stream, starting from the first byte of the header.

As illustrated in Fig., the overall header size is 32 bits (i.e. 4 octets). Some field values (e.g. *PDU Type* and *Message Length*) used in this fixed structure depend on the kind of eTDP messages sent by the network nodes.

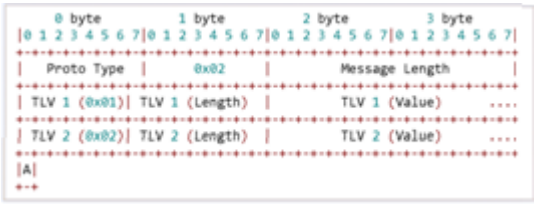



**2) Topo Request**

The topoRequest message is used by the SDN controller to initiate the topology discovery process in the network. Fig. presents the message format of a topoRequest.

Besides the header, this simple message only carries the ID corresponding to the SDN controller that sends the topoRequest message. This is the manner in which each SDN controller announces its presence to every forwarding device active in the network.

*SDN Controller ID*: Controller identifier (48 bits). The node identifier used in the messages is the MAC address. If the network controller has more than one interface, it must choose the MAC address from one of its active interfaces. This field has the




**3) Echoreply**

After receiving the topoRequest message, each network node should automatically reply with an echoReply message. This one-hop reply enables the exchange of local topology infor- mation between neighbors and the establishment of a hierar- chical control tree rooted at the SDN controllers. In Fig. 5 the frame format of an echoReply message is presented. As shown, the value in the message header (i.e. *PDU Type*), has changed to type 0x02 for indicating the echoReply message.

This message format was inspired by the use of Type- Length-Value (TLV) structures for the exchange of local neighbor information. Type-Length-Value structures have

been widely exploited by several existing standardized protocols such as LLDP. Intermediate System to Intermediate System (IS-IS) and Remote Authentication Dial-In User Service (RADIUS). among others. In this proposal we utilized TLV as an efficient method for transmitting different kinds of topology data inside the message body.

While the TLV type and length fields occupy the first two octets of the TLV format, the value field may have a fixed or variable size. In addition, it may include differ- ent types of information, containing either binary or alpha- numeric data, which is specified using the associated subtype identifiers (e.g. port component, MAC or IP address, interface name, locally assigned identifiers, etc.).

In addition, the echoReply message is used by forward- ing devices as an acknowledgment to confirm or deny the association in the control tree. In essence, each switch in the network sends an echoReply message not only to announce its topology information but to indicate its association with a specific neighbor (i.e. other switch or SDN controller). To do this, an additional association bit is included in this protocol frame.

#### 4) Topo Reply

The principal function of the topoReply message is to guarantee the proper transmission of the topology network state from the forwarding devices to the SDN controllers. To achieve this, this message format is also based on the use of TLV structures.

Fig. 6 shows a brief description of the topoReply message following the basic TLV format. This message may contain the five TLV types supported by the eTDP and presented in Table 2. The first of these (i.e. TLV Node ID), which is mandatory for every topoReply message, identifies the node that sends this message, while the others are used to provide information related to the connectivity with the node's neighbors.

Finally, we have also defined a pruning indicator in the topoReply messages. This pruning indicator is used by the nodes to notify whether they can reach the SDN con- trollers through only the neighbors receiving this notification. Clearly, nodes that have only one active port (i.e. leaf nodes) and nodes with all its downstream ports pruned (i.e. v-leaf nodes), will send the prune bit set within the topoReply message. A description of this pruning bit is given below.

*P:* Pruning Indicator (1 bit). This one-bit field enables eTDP nodes to announce to their neighbors whether they cannot provide an alternative path to the SDN controllers.

#### Protocol Operation

The presented topology discovery mechanism is initialized by each SDN controller sending a topoRequest message. This multicast message is then propagated accross the network creating a control tree topology rooted at the SDN controllers for collecting network state data. Moreover, this control tree also distributes the management of the physical infrastructure among several SDN controllers.

With the exception of the SDN controller, nodes have one of three roles, i.e. leaf, v-leaf or core, according to their position in the network topology. Leaf nodes are the nodes in the network that have only one neighbor. A node is v-leaf when it has more than one neighbor but only one of them can provide a path to the SDN controllers. The remaining switches are denoted as core nodes.

Additionally, each active port takes one of four states related to the control tree: standby, parent, child or pruned. Fig. 7 shows the port states for a given network device.

1) A standby port is an active port in the node that is not used in the control tree.
2) A parent port is an upstream port in the control tree that has first received the topoRequest message. Thus, each node has only one parent port.
3) A child port is a downstream port of the control tree that has received an echoReply message with the association bit set.
4) A pruned port is a child port that has received a topoReply message stating that it is attached to a leaf or v-leaf node.

**Algorithm 1** topoRequest Message Forwarding

```
1: Node v receives topoRequest from node u by port p
2: if node v is non-discovered then
3:     Send echoReply to node u          ▷ association bit set
4:     StateMachine (p)                  ▷ p.state = Parent
5: Send topoRequest for all ports except p
6: else
7:     Send echoReply to node u          ▷ association bit clear
8:     Discard topoRequest
9: end if
```

## 4. Conclusions

In this paper we proposed a novel protocol for discovering layer 2 infrastructures in large-scale SDN topologies. To that end, the proposed eTDP hierarchically distributes the discovery functions among switches supporting this proto- col. Unlike existing approaches, this solution enables auto- matic discovery of the network without requiring previous IP configurations or controller knowledge of the network. By using this mechanism, the SDN controller is able to dis- cover the network topology and construct a holistic network view without incurring scalability issues while taking advan- tage of the shortest control paths to each switch. Through experimental simulations with real-world topologies, we have demonstrated that eTDP provides a suitable approach for discovering the network topology with discovery times of under 0.08 ms in the three considered networks. The obtained results also show that the overall number of packets gen- erated per switch is not affected by increasing the number of SDN controllers. Moreover, eTDP achieves noticeable improvements with respect to OpenFlow-based approaches, with the most significant reductions seen in comparison to the current OFDP.

Classified by Type

| eTDP Control Frame | SDN Controllers | Network Topology | | |
|---|---|---|---|---|
| | | Atlanta | Sun | Pioro |
| topoRequest | 1 | 1,77 | 2,60 | 3,34 |
| | 2 | 1,62 | 2,46 | 3,25 |
| | 3 | 1,51 | 2,35 | 3,17 |
| | 4 | 1,44 | 2,26 | 3,10 |
| | 5 | 1,38 | 2,17 | 3,04 |
| echoReply | 1 | 1,99 | 2,82 | 3,52 |
| | 2 | 2,03 | 2,85 | 3,58 |
| | 3 | 2,04 | 2,86 | 3,61 |
| | 4 | 2,06 | 2,84 | 3,63 |
| | 5 | 2,06 | 2,81 | 3,64 |
| topoReply | 1-5 | 1,00 | 1,00 | 1,00 |
| Total | 1 | 4,75 | 6,42 | 7,86 |
| | 2 | 4,65 | 6,32 | 7,83 |
| | 3 | 4,55 | 6,21 | 7,77 |
| | 4 | 4,51 | 6,10 | 7,74 |
| | 5 | 4,43 | 5,98 | 7,67 |

## References

[1] T. Bakhshi, ‘‘State of the art and recent research advances in software defined networking,’’ *Wireless Commun. Mobile Comput.*, vol. 2017, no. 4, Jan. 2017, Art. no. 7191647.

[2] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, ‘‘Software-defined net- working: State of the art and research challenges,’’ *CoRR*, 2014. [Online]. Available: http://arxiv.org/abs/1406.0124

[3] M. Aslan and A. Matrawy, ‘‘On the impact of network state collection on the performance of SDN applications,’’ *IEEE Commun. Lett.*, vol. 20, no. 1, pp. 5–8, Jan. 2016.

[4] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, ‘‘Log- ically centralized?: State distribution trade-offs in software defined net- works,’’ in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, Helsinki, Finland, Aug. 2012, pp. 1–6.

[5] S. Khan, A. Gani, A. A. Wahab, M. Guizani, and M. K. Khan, ‘‘Topology discovery in software defined networks: Threats, taxonomy, and state- of-the-art,’’ *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 303–324, 1st Quart., 2017.

[6] POX. *Network Software Platform*. Accessed: Oct. 25, 2018. [Online]. Available: https://noxrepo.github.io/pox-doc/html/

[7] RYU. *Component-Based Software Defined Networking*. Accessed: Oct. 25, 2018. [Online]. Available: https://osrg.github.io/ryu/

[8] T. Alharbi, M. Portmann, and F. Pakzad, ‘‘The (in)security of topology discovery in software defined networks,’’ in *Proc. IEEE 40th Conf. Local Comput. Netw. (LCN)*, Oct. 2015, pp. 502–505.