IET Networks

Special Section: SDN and NFV Based 5G Heterogeneous Networks

Efficient topology discovery algorithm for software-defined networks

ISSN 2047-4954
Received on 10th April 2017
Revised 9th July 2017
Accepted on 8th August 2017
E-First on 7th September 2017
doi: 10.1049/iet-net.2017.0066
www.ietdl.org

George Tarnaras[1] ✉, *Fanouria Athanasiou*[1], *Spyros Denazis*[1]
[1]*Electrical and Computer Engineering Department, University of Patras, Rio, Patras, Greece*
✉ *E-mail: g.tarnaras@ece.upatras.gr*

Abstract: Software-defined networking (SDN) already has an established presence in the networking industry as an emerging paradigm for designing network architectures, decoupling the control from the forwarding plane and offering a centralised way of network management. Cloud and enterprise infrastructure engineers have already adopted such techniques for building up their environments, deliberating the cost and flexibility advantages gained from this technology. However, smaller companies may choose a progressive replacement of their equipment due to operational and budget constraints. Hence, mixing equipment could lead up to hybrid SDN environments making network management even more complex. In this study, an event-based generic topology discovery algorithm, capable of providing an efficient packet discovery mechanism using existing protocols in SDN networks, is proposed. For testing and evaluation purposes, address resolution protocol and link layer discovery protocol are being used to extract the information needed and reconstruct the topology to the controller. Implementation is accomplished using Internet Engineering Task Force's ForCES framework.

1 Introduction

Software-defined networking (SDN) [1, 2] and technologies associated with this network paradigm have established their presence in the networking industry during the last years; therefore, the demands in efficient traffic and network management have become an issue. Leading organisations have already adopted SDN solutions based on OpenFlow and controllers like Opendaylight, Ruy [3, 4] etc. In addition, researchers and small companies adopt SDN-based solutions to take advantage of their benefits regarding costs, flexibility etc. However, SDN devices may have to operate alongside with traditional L2 switches and this wrap has to deliver reliable services to an end user or experimenter. In addition, service orchestration tools have undertaken the role of deploying advanced and complex services with less effort than ever. In this network context, hundreds of virtual, physical and container machines may co-exist. Thus, troubleshooting and applying efficient traffic and routing policies in such an environment can turn out to be a puzzling occupation. The key point for making this information available to the SDN controller and afterwards to the administrator, facilitating the entire management procedure, is to efficiently discover the topology and the respected links between network components.

Despite the high importance of topology discovery, yet, there are no standards and protocols specified for clearly defining this process [5]. Efforts proposing novel algorithms and methods exist, but these solutions imply the development and standardisation of new protocols. However, we can exploit existing protocols already running in SDN environments for this purpose. Currently, controllers using OpenFlow [6], the most widely adopted SDN protocol, are making use of slightly modified link layer discovery protocol (LLDP) frames for performing the topology discovery process. Still, research performing in this area states that the current method carried out from OpenFlow is not optimal [7, 8]. Hence, in the near future, limitations might arise to network architectures based on such protocols and methods. In this work, a centralised topology discovery method taking advantage of existing protocols already running on the control and forwarding plane is being proposed. Our method automatically extracts the topology information and reports it to the controller without the need to modify existing protocols. This is accomplished taking into consideration the possible connectivity between SDN components with traditional ones and the protocols already running on the forwarding plane. Consequently, all information required by an SDN controller to create the topology map is obtained directly and automatically from a network device. In this paper, Internet Engineering Task Force (IETF)'s ForCES framework was used for modelling a network's device event mechanism behaviour [9]. However, the method described here is not limited to ForCES and other frameworks may also be used. For proof of concept and experimenting scopes two protocols were examined, address resolution protocol (ARP) [10], one of the most usual protocol used among L2 network devices, and LLDP [11], the protocol adopted from OpenFlow.

The remainder of this paper is structured as follows: Section 2 gives an overview to SDN and current methodologies regarding topology discovery. Section 3 details the proposed generic packet discovery algorithm, which we use for performing efficient topology discovery. Finally, Section 4 describes the implementation aspects of modelling with ForCES framework ARP and LLDP and presents result measurements. The paper sums up with conclusions and future work.

2 Topology discovery in software-defined networks

2.1 Software-defined networking (SDN)

This method mainly arose from the need to manage centrally complex networks, physical or virtual, in distributed and cloud environments [12, 13]. In a software-defined network, the forwarding plane separates from the control plane where all network's ingeniousness is kept. This way an administrator is having a centralised view of the network's topology. Hence, he can easily monitor the network's health and he is able to drive the network to a desirable state that meets various traffic and QoS demands. In the most common approach, this is realised by defining a forwarding and control plane, which are communicating through APIs [2]. In such an architecture, there is no restriction of where to deploy the controller, which normally operates in the same network with the forwarding plane. Moreover, building a flexible and dynamic network requires the controller to be able to span across multiple networks and manage numerous network devices. It means the controller should have a distinct view of the

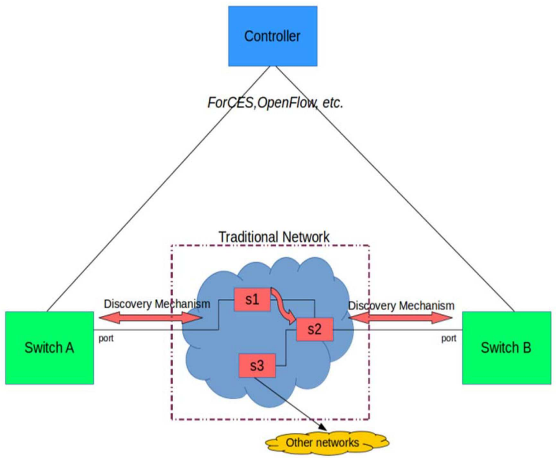

Fig. 1 *Hybrid SDN environment*

```
Begin
    initialize_system:
    start NICs;
    cleanup local_neighbors'_table;
      establish_connection to controller:port;
    detect_new_nodes:
    set ethertype_filter;
      compile filter;
    capture packets;
      extract information;
        compare with local tables;
        case: new
          update local tables;
          trigger update_event;
        case: existing
          reset alive_timer;
    continue;
End
```

Fig. 2 *Proposed topology discovery algorithm pseudo-code*

existing connectivity among the forwarding layer's devices. To this end, it is clear that topology knowledge is a key aspect in order to manage any forwarding devices in an efficient fashion.

2.2 OpenFlow and LLDP topology discovery

At the moment of writing, OpenFlow [6] uses a modified version of LLDP [11] for performing topology discovery. LLDP is a vendor-neutral, one-hop protocol operating at open systems interconnection (OSI)'s layer 2. Devices within a local area network use LLDP to advertise their capabilities.

An OpenFlow switch consists of one or more flow tables, a group table that performs packet forwarding and an OpenFlow channel to an external controller. For neighbour discovery process, it uses LLDP frames encapsulated in `packet_out` and `packet_in` commands [14]. The discovery process initiates having the controller to send out `packet_out` commands in every switch. When a switch receives a `packet_out` message, it floods all of its ports with LLDP frames. An OpenFlow switch receiving these packets will search its local tables for the incoming flow entry. Then, if a flow for this entry does not exist, it sends this packet to the controller with a `packet_in` message. This way the topology as well as the forwarding devices local tables are updated periodically.

The approach described so far is limited to the case where all network devices are OpenFlow enabled. In cases where traditional L2 switches are concurrently present in the forwarding plane, as shown in Fig. 1, the controller is not able to detect them throughout `packet_out` and `packet_in` commands, given that the controller cannot manage 'traditional' switches. Moreover, the discovery mechanism is based on a modified LLDP version, which is a one-hop protocol. Due to this hop limit, the devices cannot forward information in topologies where more than one traditional switch exists between the SDN switches. Some controllers in order to achieve this, combine protocols like LLDP with others like broadcast domain discovery protocol [15]. Nevertheless, using additional protocols just adds extra unnecessary traffic to the network, rather than using existing protocols and an automated discovery process as this paper introduces.

2.3 Hybrid SDN

As might be expected, SDN solutions, where a controller can dictate the totality of the forwarding plane's devices are not the rule (Fig. 1). It is common for network engineers to mix traditional network devices with SDN enabled ones in order to avoid a complete overhaul of a current infrastructure. This tactic leads to hybrid network environments. In such environments, direct or non-direct connections between SDN and traditional switches may exist. Directly connected devices can easily be discovered using LLDP but for indirect links, another method should be invented. In these situations, where blend traffic crosses the network, the controller shall be able to identify the connections between the forwarding devices. Then, suitable forwarding rules can be dictated to the SDN devices to get the most of the network efficiency and productivity. To accomplish this, a discovery mechanism capable of extracting information between SDN and non-SDN devices must exist. In this paper, protocols commonly running on network devices like ARP and LLDP have been used as is, without modifications, to uniquely identify SDN and traditional devices and build the relative connectivity links between them. This way the controller was able to identify connectivity links between directly and non-directly connected SDN devices.

2.4 Modelling the discovery mechanism with ForCES

IETF's ForCES defines a framework [9] and associated protocols [16] to standardise information exchange between the control and the forwarding plane. Elements that exist in the control plane are called control elements (CEs) and those in the forwarding plane forward elements (FEs). Each FE consists of one or more logical function blocks (LFBs) managed by the controller. LFBs are ForCES's basic building modules and each one implements a specific function [17]. The data-path constructs from several interconnected LFBs modelled and realised by abstracting and describing the underlying functions with XML. Thus, all essential information for establishing communication between a forward and a control element are being specified. Managing the data path between LFBs in an optimal manner requires that the controller is able to discover LFBs connections and their topology.

In previous work [18], an LLDP LFB was modelled with ForCES for performing topology discovery. In this study, the topology discovery algorithm was extended by adding an ARP LFB to be able to perform topology discovery in hybrid software-defined networks.

3 Event-based discovery algorithm

3.1 Algorithm description

Constructing the topology view requires the controller to collect the necessary information about the interconnections and the links between the existing devices in the forwarding plane. As far as topology is essential for SDN, this process should automatically initiate whenever a new switch enters the topology, without the controller's trigger.

This paper proposes the utilisation of an event-based mechanism running on every SDN switch, capable of creating events when it detects traffic generated by different, user-specified, protocols. A mechanism for listening to network traffic was implemented on every switch's available port. Whenever traffic from a predefined protocol was detected locally, an event was triggered and relative information was reported to the controller. This information provided key indicators regarding a neighbour device. A top-level abstraction of the mechanism proposed and implemented for acquiring the aforementioned information is described with the pseudocode as shown in Fig. 2.

For a node to be uniquely identified inside a topology map, the knowledge of three fields is proposed:

```
<eventeventID="2">
        <name>UpdateMap</name>
        <synopsis>Information from CE that
        new nodes are available for the
        datapath</synopsis>
        <eventTarget>
        <eventField>Map</eventField>
        </eventTarget>
        <eventChanged />
        <eventReports>
            <eventReport>
            <eventField>Map</eventField>
            <eventSubscript>Neighbors</even
            tSubscript>
      </eventReport>
      </eventReports>
</event>
```

Fig. 3 *A ForCES' model event description*

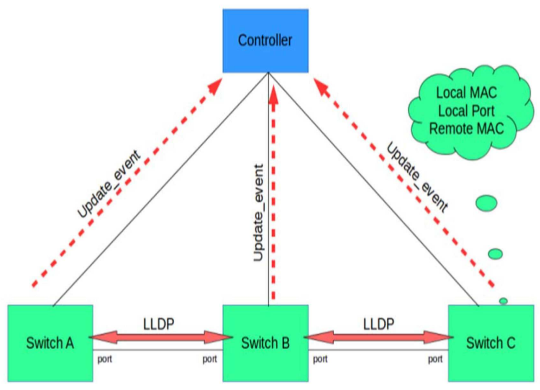

Fig. 4 *Discovering a node using LLDP*

i. The local device' MAC address.
ii. The MAC address of the remote machine to discover.
iii. The local port identifier, from which capture has been done.

Having defined these fields and reporting this kind of information back to the controller for further processing, a topology map describing the links between the devices could be extracted. Other network or analytics related functions in need of topology information could also consume these topology data. A common example of such a function could be routing. This method also solves the topology discovery problem in hybrid SDN networks, where additional protocols should be used to overcome the discovery limitations of the one-hop LLDP. The proposed technique uses various protocols, which are already running on the forwarding plane and can contribute to the topology discovery process. Hence, it is possible to define logical and physical links between non-directly and directly connected SDN devices, when traditional ones exist between them. Furthermore, the responsibility of initiating the discovery process is offloaded from the controller and now every controlled device automatically reports changes that occur. As will be explained, for proof of concept scenarios unmodified ARP and LLDP were used in this paper to supply the controller with the necessary information for performing topology discovery.

3.2 Implementation using ForCES

Through ForCES model [17], a flexible way of describing network functions is defined, utilising an XML description as the abstraction layer of an LFB. Therefore, for testing purposes in this paper, two LFBs were modelled by abstracting the necessary information and portraying the functionality of LLDP and ARP protocol correspondingly.

Leveraging ForCES events functionality, the LFBs can report mandatory information for topology discovery back to the controller. Then the controller performs further processing to these topology data to build the relations between each FE. Each event, as stated in [17], is modelled with:

- a unique 32 bit identifier,
- an LFB component, used to trigger the event, known as the event target,
- an action, sending a notification to the controller when something changes to the event's target,
- information, about what should be reported to the CE from the FE when an event triggers.

The XML code shown in Fig. 3 details a `ForCES_event`, which triggers upon a new node discovery. This event activates whenever a change happens to a switch's internal table, where all information about neighbours is saved.

This information is extracted by analysing the corresponding frames that are captured from an LFB's network interface card (NIC). For capturing these frames, the C code library `libpcap` [19] was used. Specific filters were applied to ingress traffic, based on its `ethertype` Ethernet field, to isolate frames, extract mandatory information and report it to the controller. The protocol values used in the tests are $0 \times 88\text{cc}$ for LLDP and 0×0806 for ARP, respectively, as defined in their specifications.

In order to perform the necessary tests to check the validity of the algorithm, a small-scale evaluation testbed was set up. Physical, general-purpose machines based on low-end `Core2Duo E8400 CPUs`, having `Ubuntu-Server 16.04` installed played the role of the switches. The kernel version regarding the NIC drivers used was `4.8.0-44-generic x86_64 Linux kernel build`. Additional fast-Ethernet NICs were attached in order to support enough physical connections for building up a sufficient testing topology. Required protocol daemons such as `lldpad` [20] for the Linux implementation of LLDP were also installed. The controller was set up to a different physical location and network from the forwarding plane. Thus, each forwarding device was connected to it using its public IP address and a function specific port, in our case ports `5000` for connectivity and `5001` for receiving the `events`. Following the suggested methodology, topology could be built in scenarios with directly and non-directly connected links between the network devices.

3.2.1 LLDP use case: As already mentioned, LLDP is a one-hop protocol running on the L2 of the OSI stack. A station uses this protocol for advertising its capabilities. Therefore, no modifications were required in the LLDP protocol mechanisms in order to extract the information needed for topology discovery (Fig. 4). A node is able to discover a respective neighbour whenever a new node connects to an existing topology and its LLDP agent begins sending advertising messages.

In each switch, an LLDP LFB is running and a local mechanism is inspecting ingress frames for the LLDP `ethertype`. When a suitable frame is detected from a NIC then further processing via checking the neighbour tables is done locally in the FE. If the frame received comes from a new neighbour, an `UpdateMap` event is sent to the controller to update the topology view.

3.2.2 ARP use case: The mechanism for performing topology discovery using ARP is similar to the LLDP use case, but now the protocol's exchanging frames are not one-hop and by default they do not provide any hardware and topology information. Moreover, every switch in the forwarding plane should be able to capture these kinds of broadcast frames that are being transmitted from a node. Afterwards, suitable packet filtering actions shall be defined in the modelling process. Examining the protocol's specification, a frame described as Gratuitous ARP (GARP) is specified [21]. GARP is a frame sent by a network device to declare its presence in a network. This frame is sent out as soon as this device boots or a specific link goes up or changes state. These packets can easily be traced, as `destination_IP` in a packet equals the `IP_address` of the host issuing the gratuitous ARP. Therefore, having modelled an LFB implementing proper kind of packet filtering, detecting GARP packets and sending update events to the

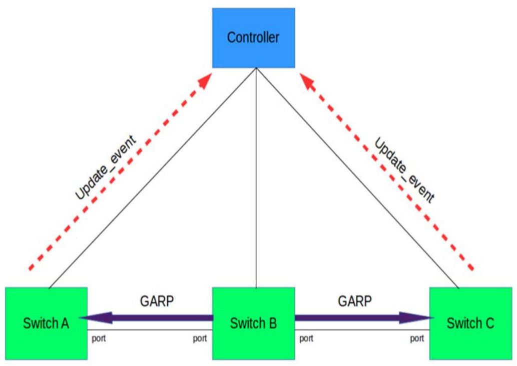

Fig. 5 *Discovering a node using ARP*

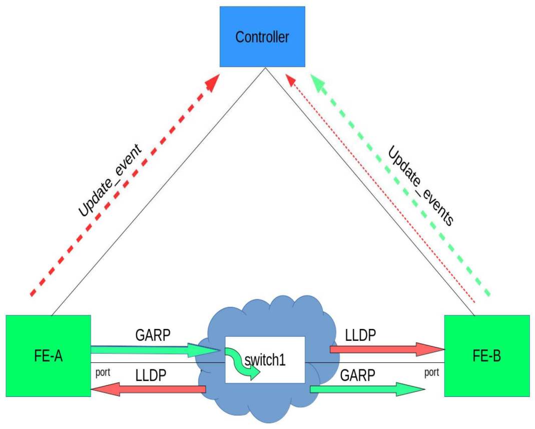

Fig. 6 *Performing discovery in hybrid environments*

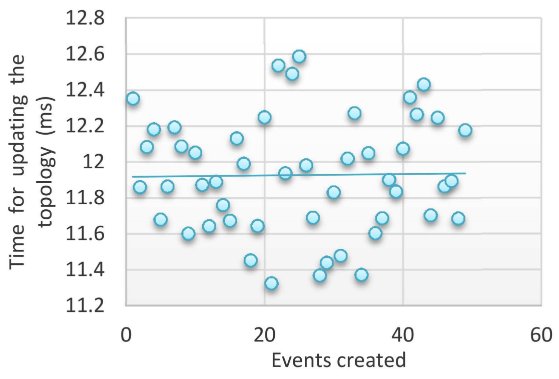

Fig. 7 *Node discovery time using LLDP*

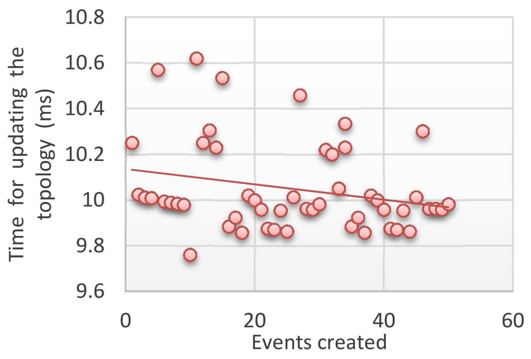

Fig. 8 *Node discovery time using ARP*

controller, ARP turned out to be an eligible topology discovery protocol.

As the proposed algorithm in this paper defines, each node's link to another node can be uniquely identified by specifying the remote `MAC_address`, the local capture port and the `MAC_address` of the NIC from which capture is happening. In ARP case, frames are sent all over the same broadcast domain network. It means the GARP events require some extra processing to match the address of the received broadcast frame with the initial device issued this frame. Hence, extracted ARP information should be cross-examined with information given from the captured Ethernet frames regarding the sender' *source MAC address*. In cases where Ethernet and ARP fields are identical in these frames, the node and the respective link can be declared as direct and unique and the controller can build up the suitable topology graph (Fig. 5).

3.2.3 Hybrid SDN use case – ARP in conjunction with LLDP: For demonstrating the proposed algorithm, additional tests executed. We were using both LLDP and ARP in a simple hybrid SDN environment, combining their existing LFB implementations. Traditional switches were installed between the SDN ones which were already running the LFBs defined in the previous use cases. The testing topology, which was successfully reported by the FEs and reconstructed back to the controller, is shown in Fig. 6.

In this use case FE-A is inserted to the topology. Then, the connectivity between the SDN switches is established through the usage of a traditional switch. Thus, issuing an LLDP frame from FE-A cannot lead this node to be discovered by FE-B. The in between switch, which cannot be managed from the controller breaks the discovery process. The controller should obtain information regarding switch1 in order to identify the connections between the elements it can manage, meaning FE-A and FE-B. The switch1 is running LLDP protocol and thus FE-A and FE-B can detect that a neighbour device exists. However, as far as switch1 cannot report anything to the controller, no further results can be exported concerning the devices' connectivity. Information needed in order to define these links is provided by GARP broadcast messages, which are detected and reported to the controller. Hence, FE-B can report events caught by the indirectly connected FE-A and the directly connected switch1. The controller having concentrated this information can perform additional filtering and matching for every device's table and define the logical connections, direct or not. Currently, similar approaches [3, 15] require the usage of extra broadcasting protocols, which could lead to excessive and superfluous usage of the network resources [7], where existing protocols could be efficiently used as demonstrated.

3.2.4 Testing results: i. *Inserting a node:* During our evaluation experiments, measurements were being performed in order to assess the time it takes for the controller to update the topology upon the insertion of a new node. Results for each case are shown in Figs. 7 and 8.

The mean time for discovering a new node was 12 ms using LLDP and 10 ms using ARP, while in a commercial controller using OpenFlow's method for discovering a node is ∼100 ms [22]. These improvements are based on our approach of dynamically performing topology discovery and directly informing the controller upon a new node insertion in the topology. This way there is no need to initiate the discovery process from the controller and every node triggers instantly an event to report a topology update. In contrast, in an OpenFlow network-based scenario an extra latency is inserted to the discovery process. This happens because the discovery packets that are sent out from the controller are being exchanged and processed from each device in the forwarding plane. This is an added latency inserted before reporting to the controller. It is worth mentioning that the results demonstrated in this paper were extracted from a small-scale evaluation testbed with low-end hardware. However, we expect similar or better efficiency in enterprise environments where production-grade hardware and software stacks will be available.

ii. *Deleting a node:* Regarding the removal of an existing node from the topology due to operational purposes or link failure, the mechanism implemented is based on the protocol's packet advertisement intervals. An interval of 60 s was set regarding LLDP. For ARP, since a GARP is detected for a new node, if no traffic relative to this node is further captured from a FE after 120 s, the node is assumed down. Afterwards, a relative event is sent to the controller to update the topology map. Different time constraints can be set to reflect more loose or strict scheduling policies.

4 Conclusions and future work

In this paper, a generic packet discovery methodology for topology creation purposes in software-defined networks was proposed. As described, this method is applicable in pure and hybrid software-defined networks capable of providing the necessary information for topology discovery. For experimenting purposes and in order to evaluate the algorithm, a ForCES based model was used to create the required LFBs and a small-scale testbed was built.

There are strong points that this mechanism can also efficiently be used in additional fields and subjects where traffic inspection is needed. Hence, our future plans include modelling the usual traffic intervals of specific protocols in various networks and implement security related mechanisms. Forwarding devices detecting such motives will be able to report to the controller these unusual traffic events and attacks like DDOS or MITM could be recognised [23]. Large-scale experiments and discovery in the context of containerised environments is also a short-term goal.

Concluding, the proposed algorithm presented in this paper details an efficient procedure on performing topology discovery, using existing protocols in multiple network environments. This approach could also be used in other applications, which require traffic information to be exchanged between network devices. Therefore, a wide range of protocols and network functions could adopt the proposed technique.

5 References

[1] Open Networking Foundation: 'Software-defined networking: the new norm for networks', April 13, 2012

[2] Haleplidis, E., Denazis, S., Pentikousis, K.*, et al.*: 'SDN layers and architectures terminology', RFC 7426, January 2015

[3] Opendaylight controller: Available at https://www.opendaylight.org

[4] Ryu controller Available at https://osrg.github.io/ryu/

[5] Ochoa Aday, L., Pastor, C.C., Fernández, A.F.: 'Current trends of topology discovery in OpenFlow-based software defined networks', 2015. Available at https://upcommons.upc.edu/handle/2117/77672

[6] McKeown, N., Anderson, T., Balakrishnan*, et al.*: 'OpenFlow: enabling innovation in campus networks', *ACM SIGCOMM Comput. Commun. Rev.*, 2008, **38**, (2), pp. 69–74

[7] Lantz, B., Heller, B., McKeown, N.: 'A network in a laptop: rapid prototyping for software-defined networks'. 9th ACM SIGCOMM Workshop on Hot Topics in Networks, 2010

[8] Mahmood, K., Chilwan, A., Østerbø, O.*, et al.*: 'Modelling of OpenFlow-based software-defined networks: the multiple node case', *IET Netw.*, 2015, **4**, (5), pp. 278–284

[9] Yang, L., Dantu, R., Anderson, T.*, et al.*: 'Forwarding and control element separation (ForCES) framework', No. RFC 3746, 2004

[10] Plummer, D.C.: 'RFC 826: an ethernet address resolution protocol', InterNet Network Working Group, 1982

[11] IEEE: 'IEEE standard 802.1 AB (LLDP), station and media access control connectivity discovery', 2009

[12] Han, B., Gopalakrishnan, V., Ji, L.*, et al.*: 'Network function virtualization: challenges and opportunities for innovations', *IEEE Commun. Mag.*, 2015, **53**, (2), pp. 90–97

[13] López, L.I.B., Caraguay, Á.L.V., Villalba, L.J.G.*, et al.*: 'Trends on virtualisation with software defined networking and network function virtualisation', *IET Netw.*, 2015, **4**, (5), pp. 255–263

[14] OpenFlow Discovery Protocol and Link Layer Discovery Protocol. Available at http://groups.geni.net/geni/wiki/OpenFlowDiscoveryProtocol

[15] Floodlight controller. Available at http://www.projectfloodlight.org/floodlight/

[16] Doria, A., Salim, J.H., Haas, R.*, et al.*: 'Rfc 5810: forwarding and control element separation (forces) protocol specification', Ie, 2010

[17] Halpern, J., Salim, H.: 'RFC5812-forwarding and control element separation(ForCES) forwarding element model', March 2010

[18] Tarnaras, G., Haleplidis, E., Denazis, S.: 'SDN and ForCES based optimal network topology discovery'. 2015 1st IEEE Conf. on Network Softwarization (NetSoft), 2015

[19] McCanne, S., Jacobson, V.: 'The BSD packet filter: a new architecture for user-level packet capture'. USENIX, 1993, vol. **93**

[20] lldpad – Link Layer Discovery Protocol (LLDP) agent daemon. Available at https://linux.die.net/man/8/lldpad

[21] Perkins, C.: 'IP mobility support for IPv4', 2002

[22] Ixia, NEC Controller Testing part 1. Available at https://www.necam.com/docs/?id=2709888a-ecfd-4157-8849-1d18144a6dda, last accessed March 2017

[23] Yu, S., Zhou, W., Doss, R.: 'Information theory based detection against network behavior mimicking DDoS attacks', *IEEE Commun. Lett.*, 2008, **12**, (4), pp. 319–321