View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Ghent University Academic Bibliography

Automatic bootstrapping of OpenFlow networks

Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet and Piet Demeester
Department of Information Technology (INTEC), Ghent University - iMinds
E-mail: {firstname.lastname}@intec.ugent.be

***Abstract*—OpenFlow decouples the control plane functionality from switches, and embeds it into one or more servers called controllers. One of the challenges of OpenFlow is to deploy a network where control and data traffic are transmitted on the same channel (in-band mode). Implementing such an in-band mode is complex, since switches have to search and establish a path to the controller (bootstrapping) through the other switches in the network. In this paper, we propose a method that facilitates this automatic bootstrapping of switches. In this method, the controller establishes its own control network through the neighbor switches that are connected to it by the OpenFlow protocol. We measure suitability of the proposed method by performing bootstrapping experiments in different types of topologies: linear, ring, star and mesh topologies. The experimental results show that the proposed method allows bootstrapping in a minimal time, which makes it suitable even for a large network.**

I. Introduction

There is sometimes need to define behavior of networks in a custom manner. Historically, this was possible only by proprietary hardware that was prohibitively expensive or impossible to obtain by researchers and experimenters. The need of this functionality exists in order to run wide-scale projects implementing new experimental protocols. Therefore, in the field of networks, the OpenFlow technology [1] which controls networks freely by software located at one or more servers (so called controllers), has caught attention of many research communities. OpenFlow is developed in a clean-slate future internet program by Stanford University, which aims to offer a programmable network to test new protocols in current Internet platforms. The core idea of OpenFlow is to decouple the control plane functionality from network switches, and to embed it into one or more servers called controllers. This makes switches/routers inexpensive. In addition, this imparts network flexibility, as the control plane functionality is moved to the controllers, while only forwarding is required to be done in hardware.

OpenFlow is based on the fact that most modern routers/switches contain a proprietary FIB (Forwarding Information Base) which is implemented in the forwarding hardware using TCAMs (Ternary Content Addressable Memory). OpenFlow provides the concept of a FlowTable that is an abstraction of the FIB. Additionally, it provides a protocol to program the FIB via adding/deleting/modifying entries in the FlowTable. This is achieved by one or more controllers that communicate with the OpenFlow switches using the OpenFlow protocol (Fig. 1). The switch/router that exposes its FlowTable through the OpenFlow protocol is called an OpenFlow switch/router.

An entry in the FlowTable consists of: (1) a set of packet fields to match with incoming packets (called as flow), (2) statistics which keep track of matching packets per flow, and (3) actions which define how packets should be processed. When a packet arrives at an OpenFlow switch, it is compared with the Flow Entries in the FlowTable. If a match is found, the actions specified in the matching entry are performed. If no match is found, the packet (a part thereof) is forwarded to the controller. Thereafter, the controller makes a decision on how to handle the packet. It may return the packet to the switch indicating the forwarding port, or it may add a Flow Entry directing the switch on how to forward packets with the same flow.

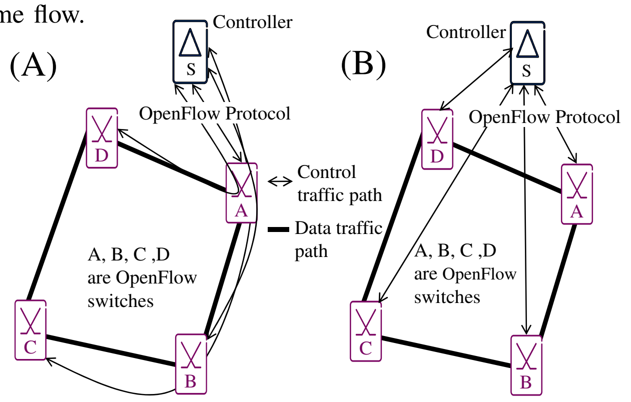

Fig. 1. OpenFlow network: (A) In-Band Mode (B) Out-of-Band Mode

In OpenFlow, control messages (e.g. messages to add Flow Entries in the switches) are required to be exchanged between the controller and switches. These messages can be exchanged either in an in-band or in an out-of-band mode. In the case of an in-band mode, control messages are sent on the same channel used to transport data traffic, whereas in the case of an out-of-band mode, control messages are sent on a different channel. As shown in Fig. 1, in the in-band mode, switches A, B, C and D share a same channel for control and data traffic, and in the out-of-band mode, switches A, B, C and D use a different channel for control and data traffic. The out-of-band mode is simpler and easier to design because the controller is directly connected (physically) to each of the switches. However, the out-of-band mode might not be possible in some scenarios, for example, a widely distributed central offices in access networks. In addition, due to the requirement of an extra physical port on each switch, the out-of-band mode is expensive to build in a real network.

In the in-band mode, switches do not need an extra physical port for control traffic. OpenFlow defines a virtual port (reserved) in a switch called as local port, which enables remote entities (e.g. controller) to interact with the switch via an OpenFlow network (in-band mode). OpenFlow, however, does not describe how control traffic paths can be established in the OpenFlow network. This task is especially challenging in the case of the in-band mode, since switches need to establish these paths through the other switches in the network. Establishing control traffic paths is important because an OpenFlow session needs to be established through these paths (bootstrapping). In this paper, we propose a method that facilitates this automatic bootstrapping of switches. In this method, the controller establishes its own control network through the switches that are connected to it by the OpenFlow protocol.

The proposed method is emulated using different types of topologies, which vary with different scales (degree, number of nodes, and distance from the controller). The emulation results show that the proposed method allows bootstrapping in a minimal time. It shows that the scalability and simplicity of the method make it suitable even for a large network.

The rest of the paper is organized as follows: section 2 presents our approach of bootstrapping, section 3 gives mathematical derivation of the bootstrapping time, section 4 describes the emulation environment and results, and finally section 5 concludes.

II. Bootstrapping of OpenFlow networks

This section is divided into three parts. The first part gives an overview of our bootstrapping approach. The second part describes the OpenFlow mechanisms and the messages that are used for bootstrapping. The third part gives the bootstrapping approach in detail.

A. *Overview of our bootstrapping approach*

Bootstrapping of a switch in an OpenFlow network requires at least two steps:

1) Assignment of connection identifiers for connecting the switch to the controller. The connection identifiers required are at least the IP address of the local port and the IP address of the controller. The other identifiers can be MAC and transport layer parameters. Transport layer parameters may include a transport layer protocol and a port number.
2) Instantiation of an OpenFlow session with the controller.

The first step can be accomplished by using protocols such as DHCP (Dynamic Host Configuration Protocol), OF-config (OpenFlow management and configuration protocol) [2] or ARP (Address Resolution Protocol). ARP can allow switches to know the MAC address of the controller. DHCP or OF-config can assign a unique IP address to a switch (i.e. local port), and can allow it to know the other identifiers (IP address of the controller and transport layer parameters) to connect with the controller.

We used ARP and DHCP to accomplish the first step. In the case of DHCP, we assumed that either the DHCP server is located in the controller node or it is the neighbor of the switch that is directly connected (physically) to the controller.

For bootstrapping, each switch runs a DHCP client and keep on flooding DHCP messages to its neighbors until it receives a reply from the DHCP server. If a neighbor is the DHCP server, it replies to the switch. In the case the neighbor is a switch connected to the controller by the OpenFlow protocol, the controller allows the switch to forward DHCP messages to the DHCP server. In the case the neighbor switch is not connected to the controller, the messages are dropped.

Once a switch has an IP address and it knows the other identifiers by the DHCP protocol, the switch learns the MAC address of the controller by the ARP protocol. If the transport layer protocol between the switch and the controller is TCP (Transmission Control Protocol), the switch then establishes a TCP connection with the controller. The switch is able to establish the connection in at least one of the following cases: (1) the controller is directly connected (physically) to the switch, (2) a neighbor switch has an OpenFlow session with the controller.

When a switch has a transport layer connection with the controller, the switch instantiates an OpenFlow session (the second step). The OpenFlow session can be established along the same path used to establish the transport layer connection. Bootstrapping of an OpenFlow network completes when each switch in the network has an OpenFlow session.

B. *OpenFlow mechanisms and messages in bootstrapping*

We used "local" and "normal" mechanisms of OpenFlow to implement the bootstrapping approach mentioned in the previous subsection. By the local mechanism, we refer to the local networking stack of an OpenFlow switch, which can be used to communicate with remote entities (DHCP server or controller). In order to communicate with remote entries, OpenFlow defines a local port. The local port allows the local networking stack to send or receive packets to or from remote entities. For bootstrapping, we run a DHCP client, a TCP/IP stack, and an OpenFlow stack in the local networking stack.

In the case of the normal mechanism, an OpenFlow switch forwards packets using Ethernet switching technologies such as MAC learning. In the case of MAC learning, MAC addresses are learned through the source MAC address and the incoming port of a packet. In the case the destination address is an unknown address or the broadcast address, the packet is flooded. In the case the destination address is already learned, the packet is forwarded through the learned port. This mechanism is used in our bootstrapping approach when a switch is not connected with the controller and the switch has to forward its own control traffic (e.g. DHCP messages or messages to instantiate an OpenFlow session) without contacting the controller.

In order to perform bootstrapping, we also used some of the messages of the OpenFlow protocol. These messages are Hello, Feature-Request, Feature-Reply, Packet-In, Packet-Out and Flow-Mod messages. With a Hello message, a switch and the controller match a version of the supported OpenFlow protocol. In the case the version matches, the controller requests the features of the switch by sending a Feature-Request message. Upon receipt of the Feature-Request message, the switch replies the controller by sending a Feature-Reply message. These messages (Hello, Feature-Request and Feature-Reply) are used to instantiate an OpenFlow session. The other messages such as Packet-In, Packet-Out and Flow-Mod messages are used to control packet forwarding in switches. In the case a switch needs to transmit an unknown packet, the packet is first sent to the controller in a Packet-In message. In the case the controller needs to send a packet through a port of a switch, the packet is sent to the switch in a Packet-Out message. In the case the controller needs to add a Flow Entry in a switch, the controller sends a Flow-Mod to the switch.

C. *Detailed bootstrapping*

In this section, we describe bootstrapping in detail by taking an example of an OpenFlow network shown in Fig. 2.

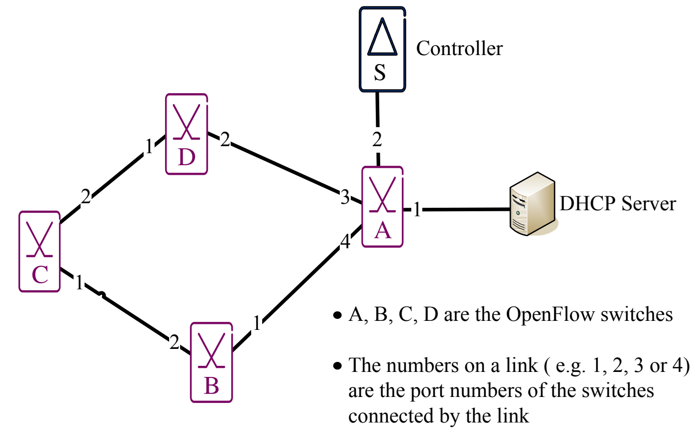
Controller
S
D
A
B
C
DHCP Server
1
2
3
4
• A, B, C, D are the OpenFlow switches
• The numbers on a link (e.g. 1, 2, 3 or 4) are the port numbers of the switches connected by the link

Fig. 2. A topology to describe bootstrapping

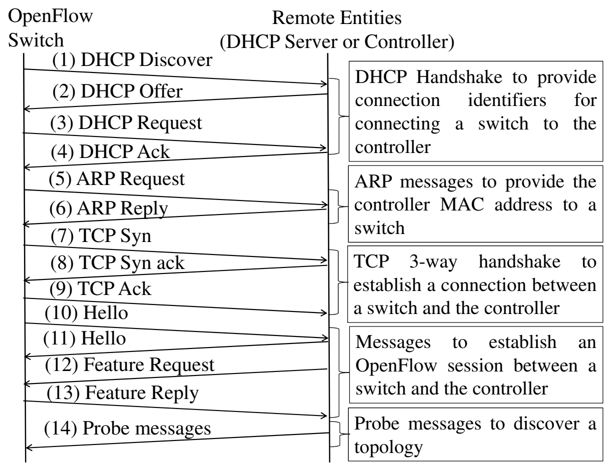
OpenFlow Switch
Remote Entities (DHCP Server or Controller)
(1) DHCP Discover
(2) DHCP Offer
(3) DHCP Request
(4) DHCP Ack
(5) ARP Request
(6) ARP Reply
(7) TCP Syn
(8) TCP Syn ack
(9) TCP Ack
(10) Hello
(11) Hello
(12) Feature Request
(13) Feature Reply
(14) Probe messages
DHCP Handshake to provide connection identifiers for connecting a switch to the controller
ARP messages to provide the controller MAC address to a switch
TCP 3-way handshake to establish a connection between a switch and the controller
Messages to establish an OpenFlow session between a switch and the controller
Probe messages to discover a topology

Fig. 3. Message Exchange between a switch and the controller

In Fig. 2, the DHCP server and the controller are directly connected (physically) with switch A. For bootstrapping, DHCP is enabled with Option 43 [3]. This option allows a programmer to program vendor-specific information in the DHCP server. In our case, vendor-specific information is the controller IP address and transport layer parameters. The DHCP clients in our bootstrapping approach request this information by sending a “vendor class identifier” in a DHCP Discover message.

The controller in our approach maintains a topology database which contains IDs (switches IDs, the controller ID and the DHCP server ID) and links information. We assigned the ID of a switch equal to the MAC address of the switch local port. At the initial stage when no switch is connected with the controller, the topology database has only the controller and the DHCP server as IDs. During bootstrapping, the controller gathers a topology of neighbor switches by transmitting a special kind of probe messages from (or to) the recently connected switch. The format of these probe messages is inline with the Link Layer Discovery Protocol (LLDP) [4]. Topology gathering is important in our approach because during bootstrapping the controller needs to find a path to any switch or to the DHCP server.

In bootstrapping, each switch and remote entities exchange a sequence of messages. This sequence is shown in Fig. 3. In order to exchange these messages, a switch uses the local and the normal mechanism of OpenFlow.

We now explain bootstrapping of switches A, B, C and D (shown in Fig. 2). The first message exchanged by each of the switches is the DHCP Discover message (shown in Fig. 3). At the stage, when the local port of a switch does not have an IP address, a DHCP client in the local networking stack transmits a DHCP Discover message from its local port.

The DHCP Discover message transmitted from the local port (switch own control traffic) is handled by the normal mechanism of the same switch. As the destination MAC address of a DHCP Discover message is the broadcast address, the normal mechanism of each switch floods the DHCP Discover message through all its outgoing ports (e.g. 1,2,3, or 4 in Fig. 2).

1) Bootstrapping of switch A: At the stage when no switch is connected with the controller, all DHCP Discover messages are dropped except the one that is transmitted through port 1 of switch A (see Fig. 2). The messages are dropped in our bootstrapping approach because neighbor switches or the controller that have received these messages have no way to forward unknown traffic. The DHCP Discover message of switch A, which is not dropped, reaches to the DHCP server. Upon receipt of the DHCP Discover message from the DHCP client of switch A, the DHCP server returns a DHCP Offer message (message (2) in Fig. 3). The DHCP Offer message contains an unleased IP address and the string containing vendor-specific information. Switch A receives this message through port 1. As the destination MAC address of the DHCP Offer message is the MAC address of the local port (switch A own control traffic), the message is handled by the normal mechanism of switch A. Thereafter, the normal mechanism forwards the message to the local port. The message reaches to the DHCP client of switch A via the local port.

The DHCP client of switch A now stores vendor-specific information from the DHCP Offer message, and responds by transmitting a DHCP Request message (message (3) in Fig. 3) through the local port. The DHCP Request from the local port is handled by the normal mechanism of switch A. The normal mechanism then floods the DHCP request message through all its ports because the destination MAC address is the broadcast address. The DHCP Request which is transmitted through port 1 of switch A, reaches to the DHCP server. The DHCP server acknowledges the DHCP Request by sending a DHCP Ack message (message (4) in Fig. 3). Upon receipt of the DHCP Ack, the normal mechanism of switch A transmits the Ack to its local port, and thereby the DHCP client receives this message. Thereafter, the DHCP client assigns the IP address to the local port. The local networking stack of switch A now parses the vendor-specific information (stored at the time of the DHCP offer message), and it knows the controller IP address and transport layer parameters to connect with the controller.

The local networking stack of switch A now transmits an ARP request message (message (5) in Fig. 3) through the local port to know the controller MAC address. The ARP request is now flooded by the normal mechanism of switch A. The controller receives this request through port 2 of switch A. Upon receipt of the ARP request, the controller returns the MAC address in an ARP reply message (message (6) in Fig. 3). Switch A receives this reply through port 2. The normal mechanism of switch A then forwards the reply to its local networking stack by sending it to the local port. In addition, MAC learning mechanism in switch A learns the output port (i.e. port 2) to reach the controller.

Assuming TCP as a transport layer protocol in the vendor specific information, the local networking stack of switch A starts a TCP connection upon receipt of the ARP reply. In this

TCP connection, the local networking stack sends a TCP syn message (message (7) in Fig. 3) from the local port of switch A. The normal mechanism of switch A then sends it through port 2 (learned port for the controller). Upon receipt of the TCP Syn, the controller sends a TCP Syn Ack (message (8) in Fig. 3) to switch A. When switch A receives the Syn Ack, the normal mechanism of switch A forwards it to the local port. The local networking stack acknowledges then the Syn-Ack by sending a TCP Ack (message (9) in Fig. 3). At this stage, switch A has a TCP connection with the controller.

After establishing the TCP connection, the OpenFlow stack in the local networking stack of switch A instantiates an OpenFlow session by sending a Hello message (message (10) in Fig. 3) to the controller through the local port. The normal mechanism then sends the Hello message via port 2. Upon receipt of the Hello message, the controller replies back with the Hello message (message (11) in Fig. 3). The controller then sends a Feature request message (message (12) in Fig. 3). Upon receipt of the Feature Request message, the OpenFlow stack of switch A sends a Feature Reply message (message (13) in Fig. 3) through the local port. The normal mechanism sends this message to the controller via port 2. The controller receives the Feature Reply message, and declares an OpenFlow session with switch A. In the Feature reply message, switch A has sent all its attributes/parameters including the MAC address of its local port. Henceforth, the controller adds the MAC address of the local port as the ID of switch A in its topology database.

At this time the controller does not know how switch A (this is the only switch present in the topology database) is connected with the controller. To know this, the controller sends a probe message to switch A. Upon receipt of the message, switch A treats this as unknown traffic, and sends this back to the controller as a Packet-In message. The Packet-In message includes the ID of switch A and the incoming port of the probe message (i.e. port 2) in its message. Upon receipt of the Packet-In message, the controller now finds that the Packet-In message is generated by switch A and there is no path in its topology database to reach from switch A to the controller. Therefore, the controller adds a link in its topology database such that switch A is connected to the controller through port 2. In order to take control over control traffic of switch A, the controller at this time may add two Flow Entries in switch A. The first entry can be for the flows containing the destination MAC address as the MAC address of the local port. The second entry can be for the flows containing the incoming port as the local port and the destination address as the controller address.

2) Bootstrapping of switches B, C and D: When switch A established a session with the controller, switch B, switch C and switch D are in the initial phase of transmitting DHCP Discover messages. As switch B and switch D in Fig. 2 are directly connected (physically) with switch A, DHCP Discover messages from switch B and switch D reach at switch A. Switch A has now an OpenFlow session with the controller. Therefore, upon receipt of the DHCP Discover messages, switch A sends these messages to the controller in Packet-In messages. Let us take the case when the DHCP Discover message from switch B reaches to switch A. The Packet-In message in this case includes the ID of switch A and the incoming port of the DHCP message i.e. port 4 in its message. Upon receipt of the Packet-In message, the controller finds that the message in the Packet-In is the DHCP message (because the message has the destination transport layer port equals to 67) and the source of the DHCP message (i.e the local port of switch B) is not present in its topology database. Therefore, the controller adds the ID of switch B in its topology database. In addition to the ID, the controller adds a link in its topology database such that switch B is connected to switch A through the incoming port of the DHCP message (i.e. port 4).

The controller does not know at this time the location of the DHCP server, therefore, it sends a Packet-Out message to switch A to flood the DHCP Discover message from all ports of switch A except the incoming port of the DHCP Discover message (port 4). Upon receipt of the DHCP Discover message from port 1 of switch A, the DHCP server sends the DHCP offer message to switch B. The DHCP Offer message is now received by switch A through port 1. However, switch A does not know how to handle this message. Therefore, it sends the message to the controller in the Packet-In message. This Packet-In message includes the incoming port of the DHCP Offer message (port 1) and ID of switch A in its message. Upon receipt of the Packet-In message, the controller calculates a path from switch A to the destination of the DHCP offer message (i.e. switch B). As the controller knows the path to switch B through port 4 of switch A, the controller sends a Packet-Out message to switch A to forward the DHCP Offer message via port 4. In addition, the controller finds that this DHCP message is from the DHCP server (because the message contains the transport port of the source as 67). Therefore, the controller adds a link in its database such that switch A is connected to the DHCP server through the incoming port of the DHCP Offer message (i.e. port 1).

Upon receipt of the DHCP offer message, the normal mechanism of switch B handles this message, and forwards this to its local port. Switch B now exchanges the other messages (the message (1) to (13) in Fig. 3) to instantiate an OpenFlow session. In this cases, all messages of switch B go through switch A.

Note that the controller at this time (the time when the switch B has exchanged all messages (1) to (13)) does not have complete information about the links of switch B. In our case, the controller does not know which port of switch B connected to port 4 of switch A. Therefore, the controller transmits probe messages through each port of switch B after having the OpenFlow session with it. A probe message contains the ID and the outgoing port of switch B from where the probe message has to be transmitted. The probe message of switch B, which is sent from port 1 of switch B, reaches to switch A through port 4. As the probe message is unknown traffic for switch A, it is sent to the controller in a Packet-In message. Upon receipt of the Packet-In message, the controller now parses the probe message, and finds that the message is sent from port 1 of switch B. As the incoming port of the Packet-In is port 4 of switch A. The controller updates the link of switch A in its topology database such that port 4 of switch A is connected to switch B by port 1. After this, the controller adds Flow Entries in switch A and switch B for the control traffic of switch B.

Like switch B, switch D in our bootstrapping approach also establishes an OpenFlow session with the controller. Like the same way, the controller transmits probe messages from switch D after having the session with it, knows the link connecting switch D to switch A, and adds Flow Entries in switch A and switch D for the control traffic of switch D.

At the time switches A, B and D have OpenFlow sessions with the controller, switch C may be in the initial stage of transmitting the DHCP Discover messages. In the case of switch C, the DHCP Discover messages are received by switch B and switch D. Switch B and switch D have now OpenFlow sessions with the controller. Therefore, the DHCP messages from switch B and switch D will be sent to the controller in Packet-In messages. Let the Packet-In message from switch B first reaches the controller. Upon receipt of the Packet-In message from switch B, the controller adds switch C in its database and adds a link in its database such that switch B is connected to switch C through port 2. In the case the controller does not know the port of switch B along the calculated path to the DHCP server (because the controller may have not transmitted/received a probe message giving information about the link between switch B and switch A), the controller replies switch B to drop the message. In the case the controller knows the port of switch B along the calculated path to the DHCP server, it replies to B to forward the message along the path.

In the case of the Packet-In message from switch D, the controller adds a link in its topology database such that switch D is connected to switch C through port 1. Like the DHCP Discover message from switch B, the controller replies switch D to forward the message to the DHCP server along the available path. Thereafter, two DHCP Discover messages from switch C may reach to the DHCP server (the one from the path C-D-A and the other from the path C-B-A). Upon receipt of these messages, the DHCP server replies to only one DHCP Discover message by sending a DHCP Offer message. Therefore, at the end one DHCP Offer message reaches to switch C. Switch C now exchanges all other messages (shown in Fig. 3) with the DHCP server and the controller. The messages exchanged are similar to the sequence of messages exchanged at the time of bootstrapping of switches A, B and D. After exchange of the messages, switch C will have an OpenFlow session with the controller.

The controller now gathers information about all links of switch C by transmitting probe messages from all ports of switch C. After this, for the control traffic of switch C, the controller adds the Flow Entries along a calculated path (shortest) from switch C to the controller.

III. Emulation environment and results

In this section, we describe the testbed, topologies, methodology and results of the bootstrapping experiments.

We performed emulation on our virtual-wall testbed which is a generic test environment for advanced network, distributive software and service evaluation. We created linear, ring, star and mesh topologies in our testbed nodes to perform boostrapping in OpenFlow networks. The topologies were created by using Linux processes in different network namespaces. In all topologies, we connected the controller and the DHCP server to one of the switches present in the topology. In the case of a star topology, we connected the controller and the DHCP server to the central switch connecting all the other switches in the star network. The number of switches connecting the central switch is varied and the effect on the bootstrapping time is shown in the results. In the case of mesh topologies, we used topologies that were developed within the COST 266 action project [6]. In this project, a basic reference topology (BT topology) and variations of the BT topology, suited for a pan-European network, were designed. The variations of the BT topology were Core Topology (CT), Large Topology (LT), Ring Topology (RT) and Triangular Topology (TT). These were obtained by varying the total number of nodes and the degree of meshedness. The CT topology and the LT topology differ with respect to the number of nodes. The BT consists of 28 nodes, the CT consists of 16 nodes and the LT consists of 37 nodes. The other derived topologies contained the same number of nodes as the BT, but the difference lies in the degree of meshedness. The maximum degree of nodes in these topologies is 7. We performed the bootstrapping experiments on all these topologies.

There are many extensions of the OpenFlow protocol. Some of the extensions have been released publicly in the form of versions. The OpenFlow 1.0 version that is developed by Stanford is called as the reference switch [7]. This reference switch contains the DHCP client software in its implementation. However, this software is abandoned in the higher versions. We integrated this DHCP client software in the OpenFlow 1.1 version (developed by Ericsson), and used this for our implementation. In addition, many OpenFlow controllers are also available for controlling OpenFlow networks. These are NOX, Beacon, Onix, Floodlight, Helios and Maestro. We implemented our bootstrapping approach in the NOX controller (developed by Ericsson [8]) and used this in our emulation.

In our emulation, the DHCP server is enabled with two options: ping checked enabled and ping check disabled. The ping check may be required to verify address availability before offering it to a client. We tested our bootstrapping approach with both the options and calculated the bootstrapping time. In the case of ping check enabled, the DHCP server pings an IP address before offering it to the DHCP client. In the case the DHCP server does not receive a reply of a ping until a certain time (1 second in our case), it offers the IP address to the client. In the case of ping check disabled, the DHCP server offers an IP address to the DHCP client without pinging the IP address. For the transmission of the DHCP Discover message, we kept the retransmission time of the DHCP Discover messages (the time if a DHCP client does not receive a reply of the DHCP Discovery message) equals to 1 second. The DHCP client in an OpenFlow switch changes this value to a random interval between 0.90 to 1.10 second.

We now show the results of the experiments performed on different topologies. Fig. 4 shows the results of the experiments performed on linear and ring topologies. The results show a linear relationship between the bootstrapping time and the shortest distance (number of hops) from the controller. In the case of ping check enabled, the bootstrapping time of switches

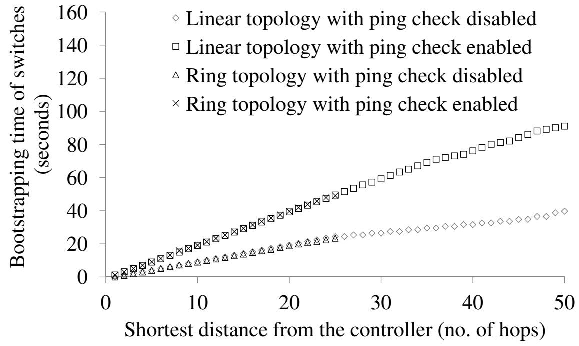

Fig. 4. Bootstrapping Experiment on linear and ring topologies

is delayed by an additional time. This is because the DHCP server waited 1 second before offering an IP address to each of the switches. In the case of the linear topology of 50 nodes, bootstrapping of all the switches took approximately 40 seconds with the ping check disabled option and 91 seconds with the ping check enabled option. In the case of the ring topology of 50 nodes, bootstrapping took approximately 23 seconds with the ping check disabled option and 50 seconds with the ping check enabled option.

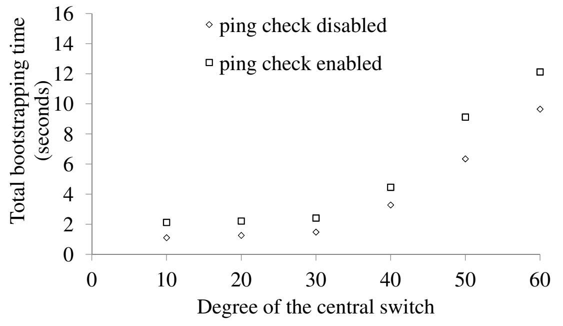

Fig. 5. Bootstrapping experiment on star topologies

Fig. 5 shows the results of the experiments performed on the star topologies. The results show that until the degree of the central switch is 30, bootstrapping took approximately 1 second in the case of the ping check disabled option and 2 seconds in the case of the ping check enabled option. After this, the bootstrapping time increases with the increased degree of the central switch. This is because as the degree of the central switch increases, more messages will be buffered in the packet-in buffer of the central switch. A message remains in the buffer until the controller responds on a forwarding decision of the message. This led to overflow of the packet-in buffer, and resulted into drop of some of the messages. In the case a DHCP Discover message drops, bootstrapping in our emulation will take additional 1 second to retransmit the next DHCP Discover message. In the case a TCP syn message drops, the TCP stack will take an additional time to retransmit the TCP syn message. This additional time increases exponentially in TCP with the number of Syn messages dropped [9].

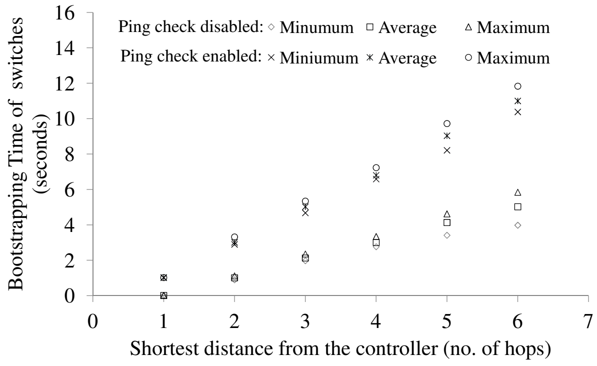

Fig. 6. Bootstrapping experiment on mesh topologies

Fig. 6 shows the results of the experiment performed on the mesh topologies developed in the COST 266 action project. The figure shows the minimum, the average, and the maximum time of bootstrapping. In all the emulated mesh topologies, we found a linear relationship between the bootstrapping time and the minimum distance from the controller. With the ping disable option, our method took approximately 6 seconds, and with the ping enable option, our method took approximately 12 seconds to bootstrap the emulated mesh networks.

IV. Conclusion

In this paper, we have proposed a method that facilitates automatic bootstrapping in an in-band case of OpenFlow networks. We have performed extensive experiments on different types of topologies, and have shown that the proposed method allows automatic bootstrapping in a minimal time. In our emulation, bootstrapping took a maximum of 12 seconds to discover the OpenFlow network created by well known pan-European topologies developed in the COST 266 action project.

In this paper, bootstrapping of OpenFlow networks is performed by using existing auto-configuration mechanisms such as DHCP. However, with the recent addition of OF-config to the OpenFlow architecture, there is an additional interface available, dedicated for configuration tasks. OF-config is based on NETCONF (network configuration protocol) [10], a transactional protocol that uses remote procedure calls on top of a secure transport channel to manage configurations on remote devices. Hence, in future work we will use OF-config, and will compare it with DHCP for auto-configuration of OpenFlow switches.

There are two topics that can enhance the work performed in this paper: (1) consideration of multi-controller networks, and (2) failure recovery in the in-band case of OpenFlow networks. In [11], [12], we performed a failure recovery experiment for the in-band case of an OpenFlow network, and achieved failure recovery within a reasonable amount time.

Acknowledgment

The research leading to these results has received funding from the EU FP7 programme under grant agreement n^o 258457 (SPARC) and n^o 258365 (OFELIA).

References

[1] N. McKeown et al., Openflow: Enabling innovation in campus networks, ACM Computer Communication Review, 2008.

[2] OF-Config: https://www.opennetworking.org/standards/of-config

[3] S. Alexander et al., DHCP Options and BOOTP Vendor Extensions, RFC 2132, 1997.

[4] IEEE standard 802.1AB: http://standards.ieee.org/getieee802/download/802.1AB-2009.pdf

[5] OpenFlow Switch Specification: Version 1.0.0 (Wire Protocol 0x01): www.openflow.org/documents/openflow-spec-v1.0.0.pdf

[6] S. D. Maesschalck et al., Pan-European Optical Transport Networks: An Availability-based Comparison, Photonic Network Communications, Vol. 5, Issue 3, pp. 203-225, 2003.

[7] OpenFlow reference switch implementation: http://www.openflow.org/

[8] Ericsson OpenFlow and NOX Controller Software: https://github.com/TrafficLab.

[9] V. Paxson et al., ”Computing TCP's Retransmission Timer”, RFC 2988, 2000.

[10] R. Enns et al., Network Configuration Protocol, RFC 6241, 2011.

[11] S. Sharma et al., Fast failure recovery for in-band OpenFlow networks, DRCN, 2013.

[12] S. Sharma et al., A demonstration of automatic bootstrapping of resilient OpenFlow networks, IFIP/IEEE Integrated Network Management Symposium (IM), 2013.