# OLC: Open-Level Control plane architecture for providing better scalability in an SDN network

Emad Alasadi, and Hamed Al-Raweshidy, *Senior Member, IEEE*

***Abstract*— The Internet has changed the world, regarding how we lead our daily lives and in recent years, new technologies, such as the internet of things (IoT) and wireless sensor networks are escalating this change. However, these technologies bring with them a rapid increase in traffic, thereby putting more load on networks. It is hard to extend the traditional fully distributed architecture and distributed aggregation mechanism to a large scale, because they suffer several drawbacks by using the data plane as a bus to transfer the control discovery messages, which increases the traffic on that plane.**

**Consequently, to solve this issue, a general architecture and discovery mechanism are introduced in this paper with Open-Level Control (OLC) plane architecture, thus providing better scalability in an SDN network. Regarding OLC, the backbone for different domains as well as the discovery process for providing a network general view are considered. OLC can scale up the network with high performance even during high traffic. In particular, it has high transparency with there being no need to change the hardware, software or protocols on the host side. Finally, the results from a 22 PC testbed verify that OLC offers a reduction in the number of discovery packets in the data plane of 84.2%, 55.2% faster discovery time and scaling up the number of subnets in an SDN network 3.2 times more than with the traditional distributed architecture and mechanism. Moreover, it provides an approximately steady rediscovery time of 4.34 secs even with very high load.**

***Index Terms*— Distributed management, Distributed-centralized management, Ethernet networks, Intra and inter domains, Scalability, Software-defined networking**

## I. Introduction

Scalability of networks is a real issue in current network architecture [1][2] owing to the rapid increase in the traffic of hosts [3], for such as video on demand as well as the growing number of end devices, in particular, in relation to development of the Internet of Things (IoT) technology [4]. Software defined networking (SDN) appears to overcome the traditional architecture issues by decoupling the control plane from the data plane to give more flexibility [5][6]. However, the standard SDN paradigm contains one controller in each network [1], which raises other issues, such as difficulties in the scalability of large networks and potential single point of failure [7]. Consequently, using multiple controllers and distributing them properly at locations in SDN architecture is an essential parameter for scaling the network [8].

In order to design an architecture/mechanism that can scale the network into a large one, whilst concurrently enhancing network performance, the following requirements should be taken into account:

- The new architecture/mechanism needs to support SDN's powerful feature, i.e. proactivity, which leads to enhancement of the response time and load balance among the network resources. That is, the general view of network is the fundamental requirement to apply proactive behaviour in an SDN for traffic manipulation [9]. As the network general view relies on the discovery process, this leads to consideration of the process as an essential one that is sensitive to the time factor. Accordingly, the discovery packets should avoid the congestion plane (i.e. the data plane) as much as possible;
- No new hardware (e.g. middleboxes) should be added to the network and no new software should be added to the host or switch sides as this could lead to downward compatibility problems;
- Standard protocols should be used to support interoperability and openness [10], regarding which, [11] fails to support this point;
- The number of protocols used for the discovery process should be as few as possible so as to avoid inconsistency, complexity and latency as a consequence of their concurrent operation;
- There needs to be support for transparency, which means users can see the system as a single one [10];
- The complexity between the intra and inter-domains should be decreased as much as possible by using the same/consistent discovery protocols. Some other designs fail to apply this, such as in [11].

No previous study has efficiently solved the scalability issue nor has completely taken into account the fundamental requirements set out above, which is the motivation behind our presenting this paper. We propose an Open-Levels Control plane architecture (OLC) to provide better scalability in an SDN network. OLC, firstly, analyses a well–known distributed mechanism, namely, the distributed aggregation mechanism, which is essential for performing the discovery process in traditional and SDN architectures. Then, novel architecture for the control plane is put forward, which defines open levels (i.e. multi-levels) of this plane with a distributed-centralized concept as well as defining the SDN switches between the control levels. In addition, an innovative dynamic discovery mechanism is introduced, which can discover multiple subnets and networks. In sum, OLC introduces full architecture and mechanisms for discovering intra and inter-

This work was supported by Brunel University London.

Emad Alasadi is with the Department of Electronic and Computer Engineering, Brunel University London, U.K. (e-mails: emad.Alasadi@brunel.ac.uk).

H. S. Al-Raweshidy is with the Department of Electronic and Computer Engineering, College of Engineering, Design and Physical Sciences, Brunel University London, UB8 3PH, U.K., (e-mail: hamed.al-raweshidy@brunel.ac.uk).

domains.

The main contributions of this paper can be summarized as follows:

- It provides practical solutions that have been ratified by extensive testbed results, which as consequence verify that OLC is better than the current distributed architecture and mechanism;
- A large testbed with 22 computers has been built to test both the fully distributed and OLC architectures/ mechanisms;
- It provides an innovative multi-subnets/networks dynamic discovery method by introducing the Dynamic Discovery Hierarchal Protocol (DHP), which provides dynamic fast discovery time in a distributed-centralized architecture;
- OLC provides superior performance in Ethernet networks, even with high load traffic;
- It demonstrates how to reduce the current network complexity by minimizing the number of discovery protocols to single one for performing the intra and inter-domains discovery process;
- It shows how to reduce significantly the resource consumption in the data plane.
- It offers means of delivering stable rediscovery time;
- The proposed model can scale the network significantly better than fully distributed architecture.

## II. Related work

Various solutions have been proposed to overcome the scalability issue since the arrival of SDN which can be categorized according to the control plane architecture, as follows.

### *A. Related work with fully distributed control plane architecture*

When designing a network that covers distributed areas, it has to be divided into multi-subnets/networks, with each having its own SDN controller. In addition, to scale a network with high performance, the proactive behaviour that is a powerful feature of SDN should be used, i.e. to install rules proactively along paths between sources and destinations, regardless of whether they are in the same subnet/network or belong to different ones. The proactive behavior of SDN relies on providing a general view of the network to each subnet/network in order to find and install routes in the routing tables between the edge devices (e.g. routers). This general view in the distributed architecture can be obtained by using a well-known discovery mechanism, i.e. a distributed discovery aggregation mechanism [11]. The Open Shortest Path First (OSPF) [12] traditional protocol is the most commonly used for this purpose for fully distributed architecture in intra-domain among subnets/networks within the same Autonomous System (AS), such as in [11].

However, there are several limitations as a consequence of using the aggregation discovery mechanism in distributed architecture, which are as follows.

- The aggregation discovery mechanism by distributing the discovery information to all subnets leads to the use one or multiple protocols to implement this mechanism, such as in the Disco model [13], where Messenger-Link Layer Discovery Protocol (M-LLDP) and Advanced Message Queuing Protocol (AMQP) are deployed to discover the network. As a consequence, this leads to an increase in the complexity of the controllers and more latency when performing the discovery. In addition, as these protocols must work synchronously and they need manual configuration, this increases the probability that the whole system will fail due to human error. **However, the proposed model only uses the dynamic discovery hierarchical protocol (DHP) which is introduced in this paper.**
- With such an aggregation discovery mechanism, the data plane is used to transfer the discovery packets through the network, which results in more load and the consumption of the resources of that plane, which consequently has an effect on the discovery and convergence time. **However, in OLC model the most of discovery packets are transferred using the control plane.**
- During peak load, the probability of failing in the discovery process for a new event (e.g. add/delete subnets) increases, because both customer data and the control discovery signal use the same plane (i.e. data plane), which can lead to congestion in the network. Consequently, the fully distributed discovery mechanism could lead to reliability issues [14], so the best discovery time with the optimum discovery path should have little or even no congestion [15]. **In our model by separating the control from the data messages the probability to fail in the discovery process decreases.**
- Aggregation of the distributed mechanism results in a number of phases (i.e. rounds) are needed to complete the whole discovery process, which in turn increases the latency of the discovery process. **In contrast, OLC uses just two fixed rounds.**
- The size of discovery packets in the aggregation models has a direct relationship with the number of subnets/networks [10], whereby the former increase when the best path becomes longer between the furthest edges (i.e. subnets) of the networks. This will be conflict with the size of the Maximum Transmission Unit (MTU) of a link that passes the discovery messages. This, in turn, leads to performing message fragmentation that is used in cases when the MTU size in less than the protocol data unit [16]. Dividing the discovery message into pieces and send them individually on the data plane leads to an increase in the probability collisions and competition, which in turn lengthens the discovery time. In addition, the limited number of available fragmentations [e.g. Intermediate System to Intermediate System (IS-IS) routing protocol which is limited to 256 fragments] leads to the inability of up scaling for large networks [16]. **However, OLC (by using centralize controller to collect discovery packets, process and separate appropriate information to each subnets) frees the size of discovery messages from the length of the best path.**

Relying on fully distributed control plane architecture, the Onix model [11] is proposed for enhancing the scalability of a network. However, it is not efficient for one with rapid changes in its conditions and states, whilst it also has the above limitations. Moreover, it uses the OSPF protocol to make the discoveries in intra-domains with no information about how this could support proactive SDN behavior. In addition, it is not sufficient for discovering inter-domains and hence, it has to rely on other models [11].

### *B. Related work with distributed architecture with a logically centralized control plane*

With this architecture, the controllers are each allocated to a single subnet as with the fully distributed architecture; however, a new top layer is defined. Firstly, this layer in some proposals, such as in [17] and [18], is used as a data store in order to be the link among the subnets' controllers and in [17] each controller can be used to control all of the network. Nevertheless, this architecture also has drawbacks, as each controller in each specified time will retrieve the full data from the data store, which will result in each having an increased cache size. In addition, each controller will increase its CPU usage and power consumption owing to it having to perform the best path calculation for the whole network. Secondly, in [17] the top layer is used also as a control channel to make connection to transfer commands between the controllers that return the architecture to the single point of failure, increase the complexity of the system and increase the response time. Thirdly, other studies, such as in [19] use the top layer as a root controller that connects directly to the local controllers, which are used as switches proxies for it. In this architecture, a specific protocol needs to be designed to connect the local controllers to the root controller, such as in [20] , which increase number of protocols that are used for discovery and hence, the synchronization among these protocols could affect the general view consistency. In addition, there is complexity in the root controller as its role is not just the discovery process, for it also has to answer the outgoing requests from that subnet/network. In more detail, the outgoing requests from the subnet pass from the local switch to the root controller, which installs rules in all local switches along the path to the edge device. This leads to increased response time as well as overhead for the root controller. As with [19], [21] and [1] use a coordinate controller in the top layer, with one controller for each domain, thereby limiting the scalability. In addition, [1] uses unified restful API between the local controllers and coordinating controller, which leads to a backwards compatibility problem as well as increased network complexity. However, there is no mechanism regarding how to discover domains and how the local controllers gather the information. Moreover, the calculation for the global path occurs in the top controller after it receives a request (i.e. not in proactive manner), which means that it neglects the most powerful feature of an SDN.

The rest of this paper is organized as follows. Firstly, in section II we discuss the related work and limiations of existing work also III we describe the current distributed mechanism and formulate the analytical paradigm. OLC is designed and its concept is explained throughout Section IV. In section V, OLC's implementation is described with algorithms, whilst in section VI, tested experiments and their results are presented. Finally, conclusions are drawn and future work building on the outcomes proposed in Section VII.

## III. DESCRIPTION OF THE DISTRIBUTED CONTROL PLANE ARCHITECTURE AND ANALYTICAL MODEL FORMULATION

In this section, we describe the distributed aggregation mechanism by analysing discovery packets in fully distributed architecture under both the current and SDN architectures. In addition, the mathematical formulation for this mechanism is calculated at the end of this section.

### *A. Distributed control plane architecture*

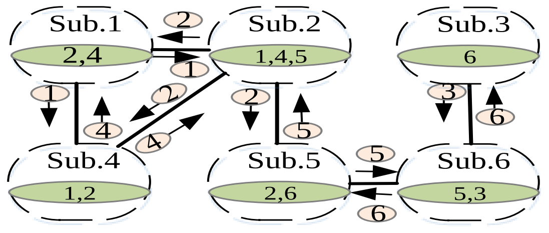
In an SDN network that covers a large area, distributed subnets interconnect each other, with each subnet having its own switches and controller. The internal switch forwards packets within the same subnet, while the edge switches work as middlebox devices (e.g. routers) to forward packets outside/inside their subnets. The controller controls every packet in its subnet depending on its policy as well as exchanging its subnet's information with other subnets in the same distributed-based network, which is why this is called a distributed control plane. This type of network normally uses the data plane bus to transfer discovery packets through the edge devices, such as in [11].

### *B. Connectivity of distributed control plane*

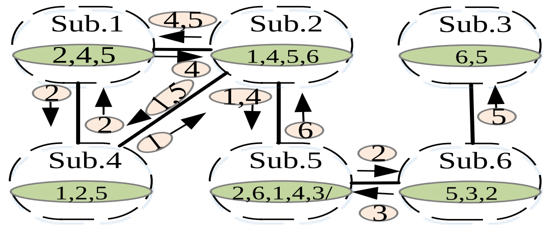
In order to make connections among subnets in same SDN network, the edge devices, such as routers/Exit_switches must exchange their information with their neighbors. In this architecture, the controller has the main script and the routing table (in the case of using a router as an edge device, then it is called virtual router [22]). The rules inside each edge device can be installed in two ways. Firstly, manually by the administrator, where he/she has to know each neighbor's information (IP address and subnet mask) in order to install a static route to it. Secondly, this can be done dynamically by using a routing protocol (e.g. OSPF and RIP), where each controller has the routing protocol's script and exchanges advertisement packets at specified times with its neighbors. After the specified discovery time each controller has an understanding of the whole network topology and installs rules in the edge devices (e.g. virtual routers) to pass packets to outside the subnet. If the virtual router is used as an edge device then it needs to refresh its connection with its neighbors by exchanging ARP packets after the specified time in order to keep the ARP table in each router updated [23], because it depends on the default gateway mechanism. Whilst if the Exit_switch is used as an edge device then this omits the use of ARP packets and there is no need to for refreshment as the Exit_switch mechanism relies on the proactive behaviour of the SDN controller through the availability of the general view of the network.

### *C. Aggregation discovery mechanism to exchange network discovery information*

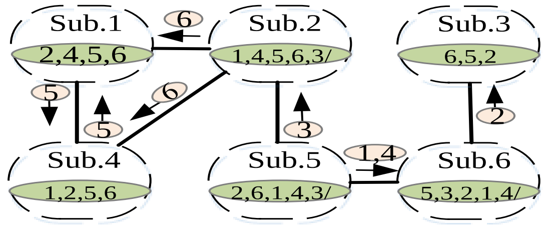
The aggregation mechanism is used in fully distributed subnets to discover the whole network's IPs and to gather statistics [11] in order for each subnet to have a consistent general network view. As a consequence, this gives SDN the powerful ability to install rules proactively and reactively in SDN switches for better performance [24]. Regarding the aggregation mechanism, we theoretically evaluate the number of phases that are needed by the network in order to let each controller in each subnet have the general view of its entirety. As can be seen in the example topology in Fig.1 (a), each controller in each subnet starts the first round by sending multicast discovery protocol's packets to all its neighbours, which is called phase1 of the discovery process. ***It should be***



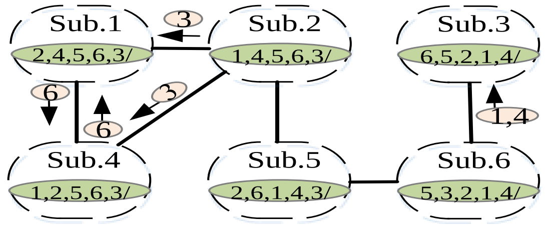
(a) Phase1



(b) Phase2



(c) Phase3



(d) Phase4

Fig.1 Shows the discovery phases when applying the aggregation mechanism in fully distributed architecture (*Note: in each phase the process on link happens before the result inside the subnets) (Note: for the figure to be not fully packed we eliminate repeated discovery messages, however in practice there is a message on each port from each subnet in each phase*)

***noted the number in the pink shape reflects the subnet number with all its discovery information involving its*** **routing and topology gathered from its neighbor subnets** [25].

As consequence of the results from phase 1, each controller just has knowledge about its next neighbors and puts this information in the neighbors' tables as well as putting the network topology in the topology table. The phase 1 results are thus:

Subnet1 just has information about subnets 2 and 4;
Subnet2 just has information about subnets 1, 4 and 5;
Subnet3 just has information about subnet 6;
Subnet4 just has information about subnets 1 and 2;
Subnet5 just has information about subnets 2 and 6;
Subnet6 just has information about subnets 3 and 5;

In phase 2, the subnets will start the second round of multicasting, as can be seen in Fig.1 (b). In this round, each controller will be used as a bridge to exchange the information among its undirected connected neighbors. In this case, the information will go one subnet further than in phase 1. As a result of phase 2, the topology and neighbors' tables will be updated, such that phase 2's results are:

Subnet1 has new information regarding subnet5;
Subnet2 has new information regarding subnet6;
Subnet3 has new information regarding subnet5;
Subnet4 has new information regarding subnet5;
Subnet5 has new information regarding subnets 1, 3 and 4 (satisfied);
Subnet6 has new information regarding subnet2;

Continuing to phase 3, the tables will be updated and the discovery packets will continue multicasting to next neighbours, which leads to the discovery information going two subnets further than in phase 1 as can be seen in Fig. 1(c), and phase 3's results are:

Subnet1 has new information regarding subnet6;
Subnet2 has new information regarding subnet3 (satisfied);
Subnet3 has new information regarding subnet2;
Subnet4 has new information regarding subnet6;
Subnet5 gains nothing new as it is in the middle in example topology, so it is satisfied first (i.e. it is first to acquire the general network view);
Subnet6 has new information regarding subnets1 and 4 (satisfied).

**In phase 4, the discovery packets will continue multicasting to next neighbours, which leads to the discovery information going three subnets further than in phase 1 as can be seen in Fig. 1(d), and phase 4's results are:**

**Subnet1 has new inf. regarding subnet3 (satisfied);**
**Subnet2 gains nothing new (satisfied);**
**Subnet3 has new inf. regarding subnet1 and 4(satisfied);**
**Subnet4 has new inf. regarding subnet3 (satisfied);**
**Subnet5 gains nothing new (satisfied);**
**Subnet6 gains nothing new (satisfied);**

**As consequence, after finishing the fourth phase, all the controllers will have the appropriate information regarding all the subnets' topology tables**.

As a result of using the distributed aggregation mechanism in traditional/SDN architectures the number of phases is equal to the best path between the furthest edges of network (i.e. furthest subnets), as in Equation 1.

$$N_{op} = B_{pfes} \qquad (1)$$

Where, $N_{op}$ denotes the number of phases and $B_{pfes}$ is the best path between the furthest subnets.

Regarding the discovery process latency, the highest controller latency refers to the time needed by the controller to multicast discovery packets, receive discovery packets and to store/retrieve information to/from the discovery tables. As the subnets work concurrently, highest controller latency is equal approximately to the latency of the slowest controller. Whilst the latency in each phase is equal to the highest controller latency plus the highest link latency, as in Equation 2.

$$L_p = H_{cl} + H_{ll} \qquad (2)$$

Where, $L_p$ denotes the latency in each phase, $H_{cl}$ is the latency of the slowest controller and $H_{ll}$ is highest link latency, which represents the slowest link in the network between subnets

Accordingly, the discovery time needed each specified time (T) is approximately equal to the number of phases multiplied by the latency of each phase, as in Equation 3.

$$D_t = N_{op} * L_p \qquad (3)$$ Where $D_t$ = discovery time

The number of packets generated in the network to complete the discovery process for one phase is equal to the summation of the number of out links from each subnet, as in Equation 4.

$$N_{pDp1} = \sum_{n=1}^{Ns} (\ N_{ol}\ )_n \quad (4)$$

Where, $N_{pDp1}$ is the number of packets generated in the network to complete the discovery process for one phase, $N_{ol}$ is the number of links from each subnet and $N_s$ represents the number of subnets

While the number of packets to complete the full discovery process is equal to the number of phases multiplied by the number of packets required to complete one phase, as in Equation 5.

$$N_{pDpF} = N_{op} * N_{pDp1} \qquad (5)$$

Where $N_{pDpF}$ represents the number of packets to complete the full discovery process

As can be seen from the equations, for a large network this requires many phases in relation to $B_{pfes}$ (Equation 1) and also an extensive number of packets in each phase, which leads to consumption of data plane bandwidth, an increase in the requirements of the control plane [11] and longer discovery/rediscovery time.

## IV. OLC DESIGN

### A. Design Goals

The OLC model is designed in this section, where a general architecture in order to enhance the discovery subnets/networks mechanisms in large Ethernet SDN networks is proposed. In addition, the dynamic discovery hierarchal protocol (DHP) for a multi-layer control plane is proposed to provide a general view of whole network, which supports SDN performing proactive behaviour.

### B. OLC units

OLC model contains several units for completing the purposes that it has been designed for, as can be seen in Fig.2. These units work with a multithread concept aimed at fast response and distributed loads on the cores of the CPUs. The Received Unit receives discovery packets from the same level,

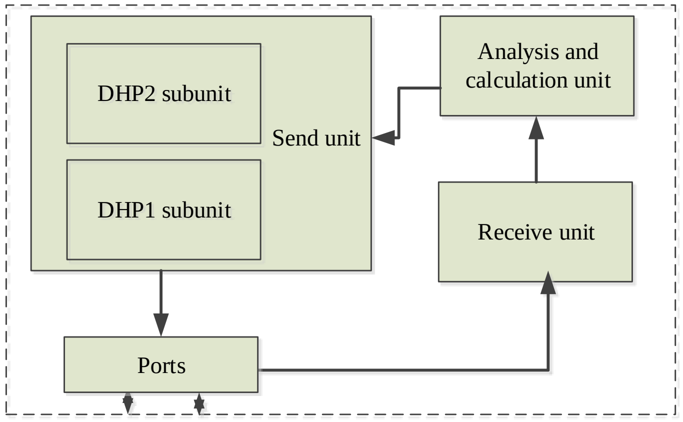


Fig. 2. OLC units in a single controller

level minus 1 (level-1) and level plus 1 (level+1) controllers, subsequently sending the messages to the Analysis and Calculation Unit that has connections with all the discovery tables. This unit will obtain, analyze and perform calculations on the received information to fill the discovery tables, including the Neighbors_topology and All_topology. Then, it sends the information to the Send Unit, which has two subunits, DHP1 and DHP2, which were assigned their names from the dynamic discovery hierarchal protocol (DHP) proposed in this paper. This unit takes its information from the discovery tables and sends discovery messages into the same/different level controllers, as is explained later in this section.

### C. General network architecture under OLC

When a network is scaled up, important requirements are a fast discovery time during bootstrap time and a fast rediscovery time for any change in the network states, such as add/remove the link between two subnets/networks. That is, the latency of the discovery time is an important factor when scaling the network, whereby if this time is low, new subnets/networks can be added and hence, the network scaled up. In order to achieve the best performance with fast discovery/rediscovery times, we believe that the centralized architecture should be combined with the distributed one.
Our proposed model involves dividing the scale concept for an SDN network into vertical and horizontal scales, where the former represents the scale of the control plane, whilst the latter pertains to that of the data plane. The ability to scale the control plane leads to scaling of the data plane, because it enhances the discovery time. As a consequence, we believe we have developed the best discovery architecture, for it combines both distributed and centralized architectures, which introduces an open-level distributed-centralized control plane architecture in an SDN network, as can be seen in Fig.3.

The vertical process in the figure pertains to the scaling up of the control plane. Regarding which, level 1 is the first level

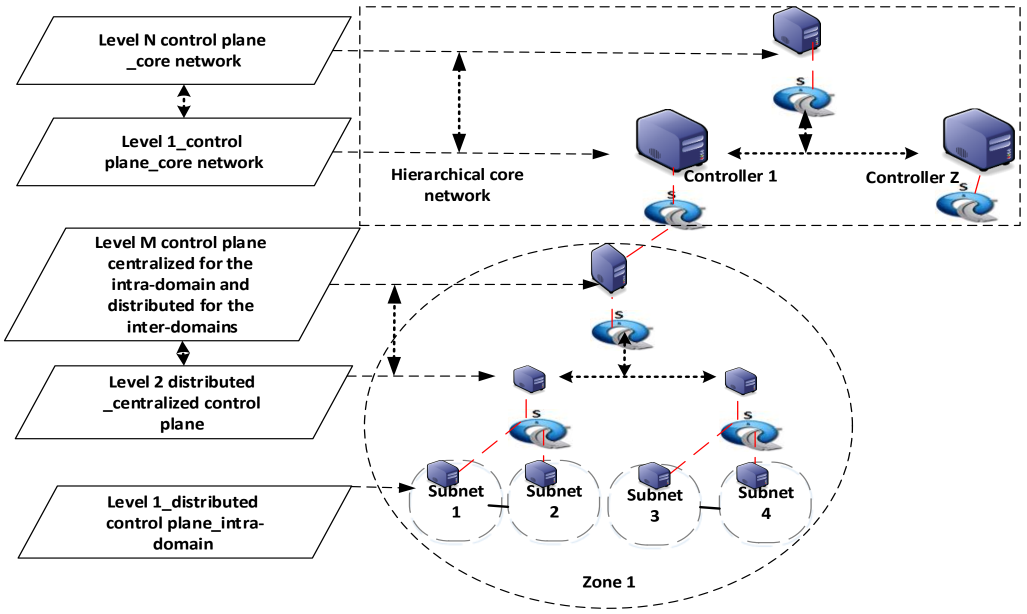

Level N control plane _core network
Level 1_control plane_core network
Level M control plane centralized for the intra-domain and distributed for the inter-domains
Level 2 distributed _centralized control plane
Level 1_distributed control plane_intra-domain
Hierarchical core network
Controller 1
Controller Z
Subnet 1
Subnet 2
Subnet 3
Subnet 4
Zone 1

Fig.3 Overall OLC architecture

in the chain and contains distributed SDN controllers, each responsible for at least one subnet, whilst the second level contains the centralized SDN controllers. Our proposed model uses SDN switches between each vertical neighbor level to give more flexibility and for recovery purposes. In order to scale the network up to a large one, such as a Metropolitan Area Network (MAN)/ Wide Area Network (WAN), the controllers in the last level of each network (i.e. level M) will represent distributed controllers for the level 1 controllers of the core network. The controllers keep connecting in a hierarchical way until those in level n are reached, which represent the top of the pyramid for all zones. The core of a network's control plane could start from level 3 or above depending on the size of network and the decision of the administrator.

On other hand, horizontally, each zone could represent a campus/ enterprise/ small city that connects to its neighboring zones using the data plane. By using this architecture, we can continue to link zones until cover a very large area, such as a country/group of cities. From the global perspective, we can imagine dividing the world into areas, with each containing one/more zones have one/more head controller(s) at the edge that can be connected in a distributed manner to exchange information.

## *D. OLC Discovery Mechanism*

As the OLC model can be scaled up to support a very large area, such as a country or even the world, there are two discovery views, with the first being with regards to the same network (intra-domains), while the other relates to a large network (inter- domains).

### 1) *Within the same network (intra-domain)*

The type of discovery we propose in this paper involves a hierarchal mechanism with M open level controllers in the intra-domains (Fig.4 shows two levels of controllers as an example). In order to perform it, the OLC model involves deploying a dynamic discovery hierarchical protocol, which is developed from the LLDP protocol. As aforementioned, this contains two elements, specifically, a distributed one (DHP1) in the controller's DHP1 subunit and a centralized one (DHP2) in its DHP2 subunit. The hierarchical discovery mechanism starts from the controllers in the subnets. Firstly, each controller in each subnet in bootstrap time will create a Neighbors_topology table. which has the fields: Neighbors_ID, Timestamps_of_packets, which are use to calculate links' latencies with neighbors and hence, identify the best paths, Edge_switch_ID, which is used to identify a subnet's edge switch and the Edge_switch_port, identifying which port is going to which subnet. As can be seen in Fig.4 (a), in each subnet the controller in level 1 during phase 1 multicasts its ID and timestamp of packet to the neighbors using one DHP1 message, while there are no messages being sent to the level 2 controller.

Each controller will receive DHP1 messages from its neighbors, which it adds to the Neighbors_topology table. The controller will perform multicasting after a specified time or if there is a change in network conditions. Secondly, the level 1 controllers will send DHP2 messages from the DHP2 subunit, which has some of the information that is in the Neighbors_topology table (i.e. Neighbors ID, link latency and Internal_subnet latency) in dictionary style, to the centralized controller in level 2, as seen in Fig.4 (b). The centralized controller in bootstrap time creates an All_topology table, which has following fields: Source_ID, Destination_ID, Link_latency and Internal_latency. The centralized controller will combine all received DHP2 messages and using the Dijkstra algorithm will find the best paths between each pair of subnets and then, will fill the All_topology table. This controller will send back DHP2 messages which contain just the crucial information to each related subnet required to install rules for reaching the destination subnets. It should be noted the DHP2's messages are different from each other, i.e. each is unique for each subnet in order to avoid sending information to unrelated one. The messages will be in dictionary format, i.e. net_X {net_Y: net_Z}, which means if subnet/network X wants to connect to subnet/network Y, it should go through subnet/network Z. The controller in level 1 will save this information in the All_view_discovery table that has been created in all controllers at all levels in bootstrap time. As a consequence, the controllers in level 1 will have a general view of the whole network. Then, the level 1 controller installs rules proactively in its switches to each destination subnet in its network relying on Edge_switch_ID and Edge_switch_port fields in the Neighbors_topology table.

Regarding the number of phases in the OLC discovery mechanism, if we assume there are two levels of controllers in the intra-domain architecture, in order to compare our architecture with the aggregation distributed mechanism in section II, each controller deals with one phase in the data plane and one in the control plane. Accordingly, there are two phases no matter how many subnets are in the network, as in Equation 6.

$$N_{op} = 2 \qquad (6) \text{ where } N_{op}\text{= number of phases}$$

Regarding the discovery time, this is needed after each specified time (T) and approximately equals the latency of the one phase from Equation 2, plus the Highest level 1 controller latency when sending/receiving DHP2 messages, plus the maximum latency of the centralized Links, which connect level 1 to level 2 controllers (there and back), plus the latency of the centralize controller (Lcc), as in Equation 7.

$$D_t = L_p + 2\, L_{cL} + L_{cc} + H_{cl} \qquad (7)$$

Where, $D_t$ denotes the discovery time, $L_p$ is the latency of one phase, $L_{cL}$ is the latency of the centralised links, $L_{cc}$ is the latency of the centralised controller and $H_{cl}$ is the highest level 1 controller latency

While the number of packets generated in the network to complete the full discovery process is equal to the sum of the number of links from each subnet and the number of links from level 1 to level 2 (i.e. number of subnets, if each subnet connects with one controller in level 2), as in Equation 8.

$$N_{pDpF} = \sum_{n=1}^{Ns} (\, N_{ol})_n + N_s \qquad (8)$$

Where, $N_{pDpF}$ represents the number of packets generated in the network to complete the full discovery process, $N_{ol}$ is the number of links from each subnet and $N_s$ is Number of subnets

2) *In the multiple networks (inter-domain)*

The OLC model provides the same mechanism as inside the network (i.e. intra-domain) to connect multiple networks in order to cover a large area, where each controller in the last level of each network will represent its network by using Network Address Translation (NAT) [26]. In addition, it will be seen in a distributed manner in relation to other controllers in the last level from other networks, as can be seen in Fig.5. Each intra-domain network will have an SDN-switch(es), which connect(s) directly to the controller in the last level of that network. That switch belongs to the data plane and is used to send information using DHP1 messages to the neighbor networks that are in different domains after applying the NAT mechanism. Whereas the DHP1 discovery messages will contain the Public_network_ID field, which represent the public IPs for that domain and Timestamp field to evaluate the link latency between two neighbor inter-domains. After receiving DHP1 messages the relevant controller will send DHP2 messages to a one level up controller (e.g. level 3) that will perform path calculation among the inter-domain networks and send back this information to the related network in a dictionary style. For example, Network_X{Network_Y: Network_Z}, which means that if network X wants to connect to network Y, it should connect first to Network Z. That information will be saved in the All_view_discovery table. The same OLC mechanism is applied when there are (n) levels of controllers covering a very large area.

## *E. Location of the controllers*

The OLC offers a flexible architecture for fulfilling different purposes. For example, if it is used on a campus/in an

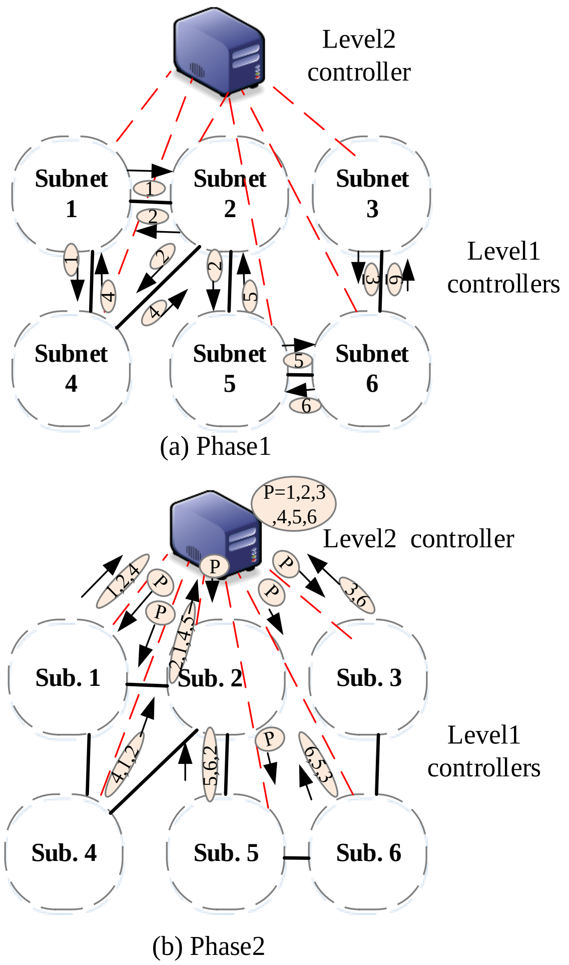


Fig. 4. Example of the OLC discovery mechanism inside one network (i.e. intra-domain) containing six subnets with two levels of controllers (P = discovery message containing route information calculated by a centralized controller for all subnets in dictionary format, i.e. Sub.x can go to Sub.y through Sub.z)

enterprise with long distances between departments then the level 1 controllers will be located near to the subnets, especially if there are many users, in order to reduce discovery/rediscovery time. While if it is used in a data center network, the level 1 controller can be located in the controllers' pool near to the last level of controllers in that network (e.g. level 2's controllers). In addition, the other controllers that represent the core network could belong to the same or different providers and could be located near to its serving zones.

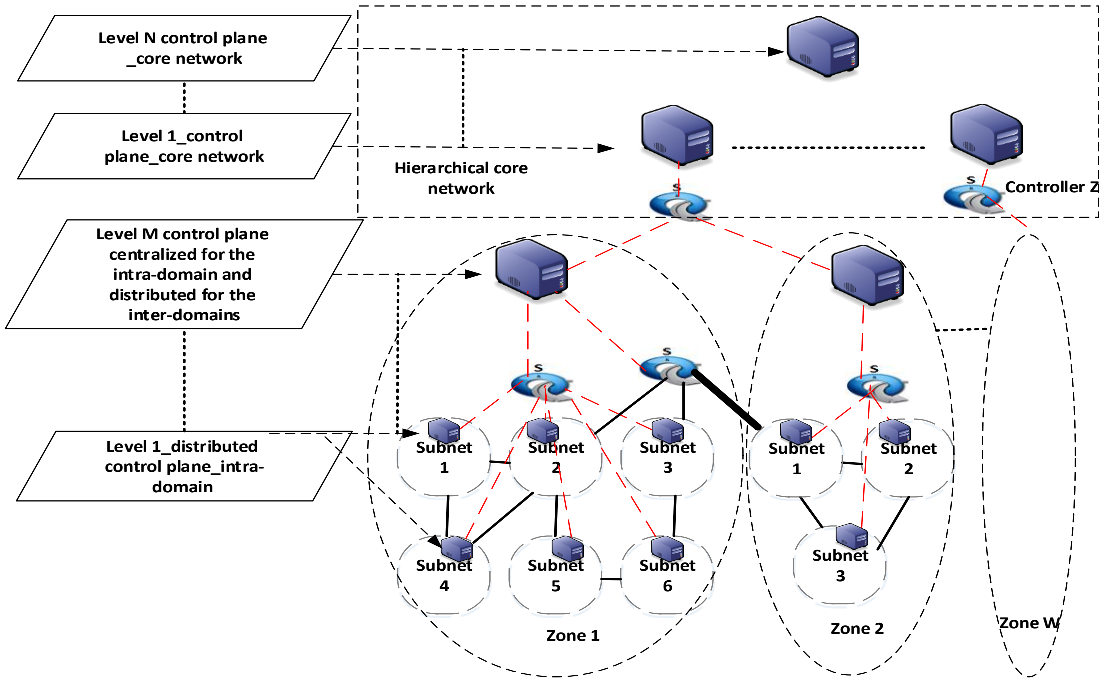

Level N control plane _core network
Level 1_control plane_core network
Level M control plane centralized for the intra-domain and distributed for the inter-domains
Level 1_distributed control plane_intra-domain
Hierarchical core network
Controller Z
Subnet 1
Subnet 2
Subnet 3
Subnet 4
Subnet 5
Subnet 6
Zone 1
Subnet 1
Subnet 2
Subnet 3
Zone 2
Zone W

Fig.5. Open-levels OLC intra and inter domains

*F. Reacting when the network fails*

There are different types of failure can happen in any network that could lead to the whole network grinding to a halt. OLC take different actions to overcome these failures and their consequences as follows.

*1) Handling level 1 controller failure*

OLC uses the standard master-slave mechanism offered by the Openflow protocol [27]. With this mechanism, the level x+1 controller works as a slave controller for the level x controllers (i.e. masters), where if any master controller related to a subnet fails, then the slave controller will take the responsibility of controlling that subnet.

*2) Handling levels 2 to n controller failure*

If a centralized controller in levels 2 to n fails, OLC provides a recovery feature by using the SDN switches in the control plane such that two or more controllers in the same level are connected to the same switch, so if the master fails the slave can serve the network. In addition, by using the same mechanism the load balance can be achieved among different controllers in same level, if they are serving the same zone/area.

*3) Handling failed links*

In the OLC architecture, more than one SDN switch could be used in the same level of the control plane to provide dependent links for recovery purposes. In addition, these links can also be used for load balancing purposes during peak control signals load.

*F. Handling subnet/network discovery (join, leave)*

Since SDN has to complete its function as a proactive installer of rules in devices along the path between the sources and destinations, it needs a dynamic fast subnet discovery mechanism to give it a general view of all subnet information. In addition, it should have a fast rediscovery time for covering any changes in the network, such as a new subnet joining or one leaving.

If a new subnet joins the network, the level 1 controllers in that subnet will start multicasting to all linked neighbor subnets, whilst simultaneously receiving DHP1s from them and then sending a DHP2 to the centralized controller in level+1 in order to get back the related general view. If a subnet leaves the network, the centralized controller will detect this through periodically monitoring the Still_alive field in the neighbors_discovery table. As a consequence of no activity from a subnet being for a specified time, a 0 will put in the Still_alive field. The centralized controller will check that field before send back the DHP2 to the relevant level 1's controller. If the Boolean value in that field is equal to 1, the DHP2 will contain the related subnet information, whilst if it is 0 the centralized controller will delete that subnet from the evaluation. Through the same mechanism, the controllers in the core network can detect the join and leave network in inter-domain networks by monitoring the activity of the edge controllers in them.

## V. IMPLEMENTATION OF OLC

In this section, we explain our OLC implementation for an open-level control plane in SDN networks in detail in relation to dynamic discovery in order to provide general view for single and multiple SDN networks covering a large area, as an addition to Ryu's [28] controller using an OpenVswitch (OVS) [29]. All the requirements set out in section I are met by OLC, which implements the DHP, thereby providing the controllers with a general view of all destination subnets/networks. It has been developed from the LLDP protocol, with DHP being the new feature. While the LLDP standard protocol just discovers the SDN switches inside one subnet, our proposal has the ability to discover all the subnets in the same network (i.e. intra-domain network), whilst also discovering other networks in different areas (i.e. inter-domain networks). To do so, the DHP has two parts as follows.

**Algorithm 1.** Distributed discovery for neighbors' subnets/networks in the same level

```
Input: Local_IP , Local_Mask, level_of_contoller, Network_public_IP, Public Mask
Output: DHP1 packets with Local_Subnet_ID/Public_network_ID, Timestamp,
        Updating Neighbors_Discovery table
1: START
2: Declare General_ID Timestamp, Type_of_DHP
3: Get controller IP , Mask and Level_of_contoller from the system configuration
4: IF controller IP = Network_public_IP then  # use NAT
5:     General_ID ← Calculate (Network_public_IPs AND Public Mask)
6: ELSE
7:     General_ID ← Calculate (Subnet_IP AND subnet_Mask)
8: END IF
9: Type_of_DHP ←0       #To send DHP1s to the same level controllers
10: REPEAT
11:    Generate DHP1 packets with General_ID and Type_of_DHP
12:    Multicast/unicast the packets from each output port in each switch
13:    Listening to Packet_in to catch DHP1 packets
14:    Decapsulating each DHP1 packet
15:    Read Message_General_ID, Type_of_DHP from each decapsulated packet
16:    IF Message_General_ID ≠ General_ID AND Type_of_DHP = 0 then
           # Means the message came from a neighbor in the same level
17:        Update Neighbors_Discovery table
18:    END IF
19:    Wait the specified time or wait for any change in subnet conditions in reactive manner
20: UNTIL terminated by the administrator
```

### A. *Implementation of the DHP distributed part in the DHP1 subunit*

This part of the protocol is located in any level of controllers in the OLC architecture that are connected to their same level neighbor controllers using SDN switches, being called the distributed part of the DHP protocol (DHP1). For example, during the bootstrap time the level 1 controllers will use this part of the DHP protocol in order to carry its own information to all neighbor controllers in different subnets in a distributed manner. Concurrently, so as to know which SDN switch connects the subnet to the other subnets, the controller monitors all the local switches using Packet_in messages. If the Packet_in message is a DHP1 message and has an ID different to the local subnet's controller ID, then OLC will register the SDN switch which enters that DHP1 as an edge switch and put Switch_ID, Switch_port, Source_subnet_ID and the Timestamp of the message in the Neighbors_Discovery table. In order to implement the DHP1 piece, we define in the DHP protocol a new type-length-value (TLV) with number 124 class and two subclasses named Type_of_DHP and Subnet_ID. Whilst The Type_of_DHP is equal to 0 for discovery messages sent at the same level, the value of the Subnet_ID subclass can be calculated by performing an AND operation between the subnet IP and subnet Mask. Subsequently, the OLC model will create DHP1 messages and multicast them to all neighbors. As a consequence, the DHP helps the destination controller to know to which source controller it is connected with. Each controller that connects to its same level neighbor controller using an SDN switch has to use the DHP1 piece (e.g. level 1 controller). It will repeat this listening and sending after the specified time or reverts to reactive mode when there are changes in subnet states, such as adding a new edge switch. We implement algorithm 1 to show how these steps are carried out in the OLC model.

### B. *Implementation of the DHP centralized part in the DHP2 subunit*

The centralized part (DHP2) located in the controllers in level 2 and core controllers (levels 3 to n). There are two roles that can be performed using this piece of the protocol depending on the sent packet direction. The DHP2 subunit sends discovery packets to the controller one level up in order to calculate the best routes and then sends back the calculated information to the level -1 controllers so as to get a full view of other subnets/networks for proactive SDN behavior. The Type_of_DHP in DHP2 packets has the value 1, if the packet is sent up to level+1 and -1, if sent down to level-1. A new subclass (net_inf.) adds to the DHP protocol in part 2. which has subnet/network information in a dictionary style (net_X:{net_Y:delay_Z}). Another subclass (Internal_latency) is added, which stores the internal latency for the subnets/networks that help in evaluating the best paths. Then, the controller in level+1 will collect all DHP2 packets from all level-1 controllers and waits for a specified time to let all join and send their information to it. After this, the controller in level+1 uses the Dijkstra algorithm to evaluate the best paths from each subnet/network to others depending on the Internal_latency field and delay between the subnets/networks. Then, it will save this information in dictionary format and send it with two types of DHP2 packets, the first of which having Type_of_DHP equal +1 for a one level up controller in order to inform the core network about the network information. While the second type, with value -1, are sent to all controllers in level-1, which will save them in the

All_view_discovery table. The centralized controllers will repeat this procedure individually according to changes in network/subnet states. The full details for the centralized part can be seen in algorithm 2.

**Algorithm 2.** Centralized discovery for the whole subnets/networks (level-1 and level+1)

```
Input: Local_IP, Local Mask, Level_of_contoller, Neighbors_
       Discovery table, DHP2 messages from level-1,
       Network_public_IPs, Public Mask
Output: DHP2 packets to level+1 and level-1
        Updating All_topology and All_view_discovery tables
1: START
2: Declare General_ID, Type_of_DHP
3: Get Controller IP, Mask and Level_of_contoller from the system
       configuration
4: IF Controller IP = Network_public_IP then  # use NAT
5:     General_ID ← Calculate (Network_public_IPs AND Public
                              Mask)
6: ELSE
7:     General_ID ← Calculate (Subnet_IP AND subnet_Mask)
8: END IF
9: Type_of_DHP ← +1 or -1
10: REPEAT
    #///////////////////////////////////start send DHP2 one level up
11:   IF  Level_of_controller = 1 then
12:         Generate DHP2 packet with (General_ID, Type_of_DHP,
                 Inf. from Neighbors_discovery table
13:   ELSE  #means all other levels
14:         Generate DHP2 packet with (General_ID, Type_of_DHP,
                 Inf. from All_topology table
15:   END IF
16:   Unicast the packet to the level+1 centralized controller.
    #//////////////////////////////////////////////////End of send one level up
    #/////////////////////////////Start listening to catch DHP2 from up and
                 down levels
17:   Listening to a specified port to catch DHP2 packet from
                 level+1(back), -1
18:   Decapsulating each DHP2 packet
19:   Update All_view_discovery table
20:   Apply Dijkstra algorithm to find best paths
21:   Generate DHP2 packet with (General_ID/ Local_Subnet_ID,
                 Type_of_DHP, Inf. from All_topology table
                 to level+1 and All_view_discovery to level-
                 1)
22:   Unicast generated packet to the level (+1 and -1) controllers
23:   Wait the specified time or wait for any change in any
          subnet/network's status in a reactive manner

24: UNTIL terminated by the administrator
```

## VI. EXPERIMENTAL RESULTS

In this section, an extensive number of testbed experiments are performed in four scenarios, with the results being presented to show the effectiveness of our proposed model. In the experiments, open source SDN Ryu is used as an Openflow controller and OVS as an OpenFlow switch. Both software are installed under Ubuntu 14.04 on 22 computers, with two of these PCs having the specifications of core i7, 3.4 GHz and 3.8 GiB memory, whilst the other 20 have specifications of Core 2 Quad, 2.66 GHz and 2.0 GiB memory. The connections between computers are made using Ethernet cables of different lengths of 1, 2 and 3 meters, with LAN cards of 1,000 Megabits per second being used in each computer; the testbed environment can be seen in Fig. 6.

Fig. 6. Built testbed with 22 computers

The OLC components are implemented on Ryu. In order to approve our proposed model's performance and its suitability for large Ethernet networks four different scenarios are performed, all of them using linear topology because they rely on the number of hops. In the OLC experiments, two levels of controllers are used while in the fully distributed architectures one level is deployed. [Each sub-experiment (i.e. result) is repeated five times and the average is taken, with the total number of runs of the testbed being 200 (i.e. 4 experiments*10 sub-experiments (i.e. results)*5 times).

The four experimental scenarios are designed as follows:

- The first scenario is performed to measure the initial system discovery time for verifying the scalability and performance of OLC compared with fully distributed control plane architectures;
- The second scenario is run to measure the rediscovery time during no load on the network under both OLC and fully distributed control plane architectures;
- The third scenario is designed to evaluate the rediscovery time under load for both OLC and fully distributed control plane architectures;
- The fourth scenario is performed by increasing the number of subnets with the aim of evaluating the number of packets that are generated as a consequence in the data plane.

### *A. Initial system discovery time: comparison between OLC and fully distributed control plane architectures*

Each system, when run from the shutdown state, takes time to reach steady state, called the boot time or bootstrap time. When connecting multiple subnets/networks it is important to measure this time for configuring the devices of the network and knowing when a steady state has been reached, such that services can be offered to the customers. A linear topology is used with an increasing number of subnets from 2 to 10, each having one SDN controller and one SDN switch. This experiment is deployed to evaluate the bootstrap discovery time under fully distributed control plane SDN architecture and our open-level control plane SDN architecture. All the controllers and the switches are timed to work concurrently and we record the discovery time needed by each subnet to have a general view of the whole network. The worst (i.e. highest) time taken is usually by the edge subnet in the linear topology.

The fully distributed model discovery time statistics provided in Fig. 7, show that when increasing the number of subnets from 2 to 10, the discovery time increases from 6.13 secs to 14.55 secs, which means it increases by 8.42 secs and

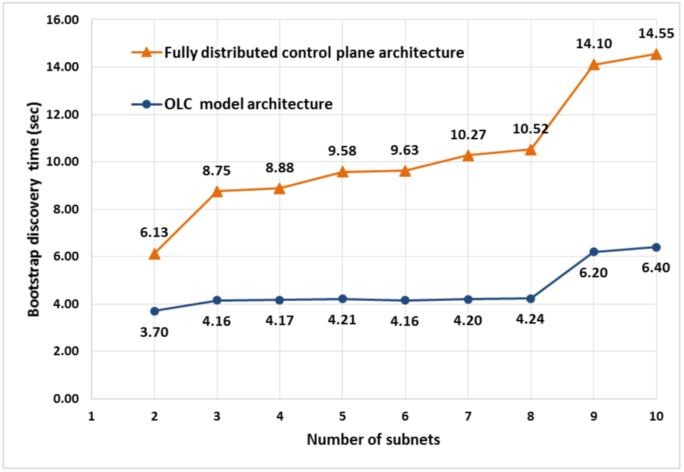


Fig. 7. Bootstrap discovery time under the OLC and fully distributed aggregation mechanisms

the average is 10.27 secs. **These experimental results are almost identical to the theoretical findings in Equation 3 (e.g. when the traditional architecture deals with 8 subnets, the bootstrap discovery time is 10.01 secs theoretically and it is 10.52 secs experimentally)**. This trend occurs because each controller in each subnet needs to send multiple discovery packets during multiple phases through the network data plane which consumes time, as explained in detail in section II. In addition, the phases are also overlapping with each other, which leads to the generation of more packets in the same link from both sides. As a result, there is congestion and an accompanying increase the time that needed to pass the information of each subnet to the neighbor subnets.

However, under the OLC model the discovery time only increases from 3.70 secs to 6.40 secs when the number of subnets is increase from 2 to 10 subnets. **Moreover, the average bootstrap discovery time using OLC is 4.6 secs, which is nearly the same as the theoretical findings in Equation 7 (i.e. 4.8 secs)**. This, in turn, means the OLC discovery mechanism can discover a network that contains 2 to 10 subnets approximately 55.2% faster than the fully distributed aggregation mechanism.

The OLC model has the ability to discover at this speed, because it has multi-level control plane architecture, which leads to the allocation of a different control plane for different discovery phases. Specifically, the next neighbors are discovered within the first phase, which in this experiment have been allocated to the level1 controllers, while the other phase is allocated to the level2 controllers, which have centralized architecture. As a consequence, the network under the OLC model can scale to 32 subnets within the same discovery time (14.55 secs) that is needed by fully distributed discovery architecture for discovering 10 subnets. This means that OLC scales the network 3.2 times more than the fully distributed discovery architecture.

### *B. Rediscovery time without load: comparison between OLC and fully distributed control plane architectures*

In this experiment, linear topology is used and the time needed for rediscovery is calculated, firstly, to detect a new event (e.g. add a new subnet to the edge of the network) and secondly, to distribute that new information to the whole network's subnets in order to update their switching tables.

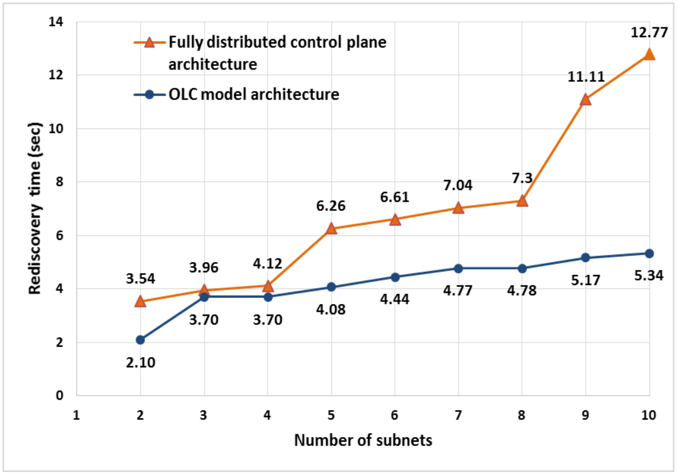


Fig. 8. Rediscovery time for network events during no load on the network under the OLC and fully distributed aggregation mechanisms

This experiment is deployed under fully distributed control plane SDN architecture and OLC architecture, as can be seen in Fig. 8.

Regarding the statistics of the fully distributed architecture, it can be seen that when increasing the number of subnets from 2 to 10, the rediscovery time will increase by 9.23 secs with two different leaps (leap to 6.26 secs from 4.12 secs and to 11.11 secs from 7.3 secs). The rediscovery time in the fully distributed architecture has this trend because when adding a new subnet to the edge of the linear topology, the rediscovery time that is needed by furthest subnets will be impacted by the number of phases (Nop) to get the new added subnet's information multiple by the Latency of each phase (LP), as described in Equation 3 in section II. As a consequence, we can expect more delay when we scale the network.

Regarding the statistics of the network under OLC, the rediscovery time is slightly increased from 3.7 secs to 5.34 secs, i.e. by 1.64 secs and with one small leap from 2.1 secs to 3.7 secs. This is because the level1 controllers just perform one distributed phase with their neighbor subnets, which has the most impact on the rediscovery time, then the remaining time is consumed by level2's controllers to multicast/unicast the switching tables to all the subnets. In addition, because OLC uses separated open-level control planes there will be no congestion on data plane links, which enhances the rediscovery time and this is opposite to the behavior of the fully distributed architecture.

## *C. The efficiency during load: comparison between OLC and fully distributed control plane architectures*

The efficiency and effectiveness of the OLC model during different load rates are presented in this experiment, where the rediscovery time has been evaluated during a range of (200-33,333,333) requests per seconds (RPS). Three fix subnets are connected together using a linear topology and then we generate loads from 20 virtual hosts on the link between

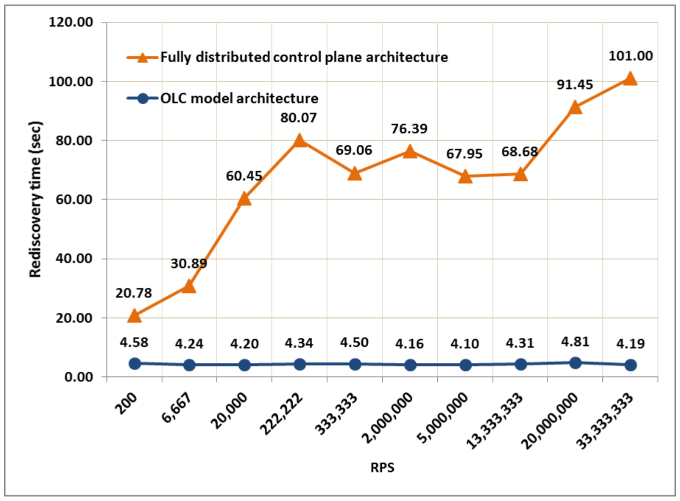


Fig. 9 Models' efficiency during load

subnets 2 and 3 in order to make congestion on that link. Subsequently, a new subnet is added to the third subnet and the time needed by all the other subnets to discover that event is recorded.

Regarding the fully distributed architecture evaluation, Fig. 9 shows that with an increase in the load from 200 to 222,222 RPS the rediscovery time is increased significantly from 20.78 secs to 80.07 secs. After that, it fluctuates with an average of 70.5 secs for loads between 333,333-13,333,333 RPS and then rises notably to reach 101 secs for 33,333,333 RPS. This trend occurs because the increase in the number of requests per second on the link between the second and third switch leads to more collisions and competition to use that link, which results in more congestion that impedes the discovery packets passing link and hence, causes a delay in rediscovery time.

In contrast, under the same circumstances, the OLC provides efficiency during different load rates, offering approximately a steady rediscovery time with an average of 4.34 secs. The reason behind this is that just one phase needs to be performed on the congested link in the data plane, which means that only one packet from each connected subnet passing that link is enough to let all the other subnets know about any new events. In addition, the centralized controller plays a big role in terms of multicasting/unicasting any changes so as to update the whole network. As a consequence, when comparing the averages of both models the proposed model has 93.5% efficiency enhancement than the full distributed architecture by reducing the response time.

## *D. Data plane bandwidth consumption: comparison between the OLC and fully distributed control plane architectures*

The bandwidth consumption from the discovery process is evaluated using the number of discovery phases and number

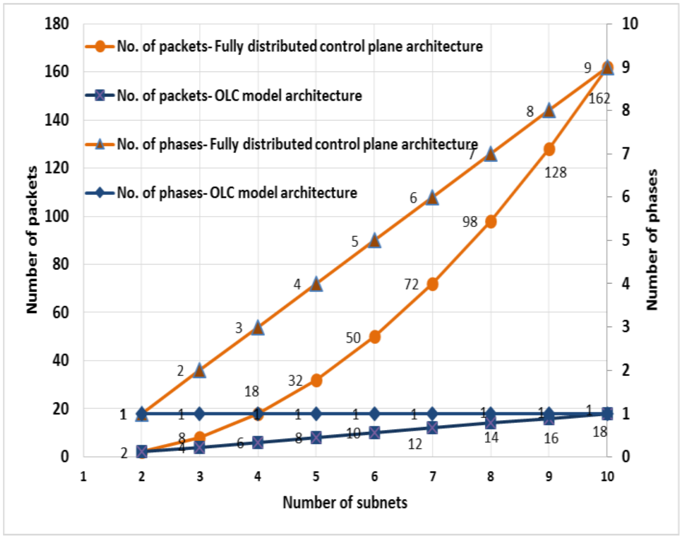


Fig. 10. Number of discovery packets and phases generated under the OLC and fully distributed aggregation mechanisms

of discovery packets, which are generated on the data plane to provide all the subnets a general view of the whole network. A linear topology is used whilst increasing the number of subnets from 2 to 10 and the Wireshark tool is used to evaluate the number of discovery packets and how many phases are needed to complete one discovery process.

Regarding the discovery process statistics under distributed architecture, as can be seen in Fig.10, when increasing in the number of subnets from two to 10 the number of phases is increased linearly from one to nine phases, which is identical with our theoretical finding in section II, where this is equal to the best path between the furthest apart subnets. As a consequence of using the linear topology, the number of phases is equal to the number of subnets minus 1. In addition, the number of discovery packets in the distributed architecture generated in the data plane increases exponentially from two to 162 packets. This is because approximately the same number of packets are generated per each phase in the network in order to complete the whole discovery process. This, in turn, leads to more congestion on the links, which increases when the network is scaled up.

On other hand, the OLC model generates just one fix phase in the data plane without any effect due an increase in the number of subnets. This is because it relies on a separated open-level control plane architecture, where the centralized controller performs the second phase discovery of the network. Regarding the number of discovery packets in the data plane, this increases by a rate of two for every new subnet added to the network. This is because every new subnet sends and receives one discovery packet with its neighbor in linear topology, if it is at the edge of the network. As a consequence of all of the above, in this experiment, the OLC model, on average, reduces bandwidth consumption by about 84.2% more than the fully distributed discovery architecture. As the data plane is an important part of the network for transferring these data among the subnets, it is essential to decrease the load on that plane [15], which can be achieved by using our proposed model.

## VII. CONCLUSION AND FUTURE WORK

In this paper, the limitations of current and recently proposed architectures and discovery mechanisms have been studied in order to provide a network general view, which in turn, supports proactive behaviour of SDN. Subsequently, the SDN based OLC architecture and implementation have been introduced to perform a general view discovery process taking into account all the fundamental requirements. By implementing an actual testbed and after an extensive set of experiments, the results have demonstrated that our proposed architecture has 93.5% better performance with 55.2% faster discovery time and can scale up the SDN network 3.2 times more than the current fully distributed mechanism. In our future work, we plan to connect OLC to the Internet to check its validity for dealing with real daily traffic. In addition, our aim is to implement a core network prototype using the OLC architecture and to test it by applying it across several virtual campus networks.

## References

[1] J. Wang, G. Shou, Y. Hu, and Z. Guo, “A Multi-Domain SDN Scalability Architecture Implementation Based on the Coordinate Controller,” in *2016 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, 2016, pp. 494–499.

[2] T. Mizrahi, E. Saat, and Y. Moses, “Timed Consistent Network Updates in Software-Defined Networks,” *IEEE/ACM Trans. Netw.*, vol. 24, no. 6, pp. 3412–3425, 2016.

[3] D. Cotroneo, R. Natella, and S. Rosiello, “NFV-Throttle: An Overload Control Framework for Network Function Virtualization,” *IEEE Trans. Netw. Serv. Manag.*, vol. 14, no. 4, pp. 949–963, Dec. 2017.

[4] S. Ray, Y. Jin, and A. Raychowdhury, “The Changing Computing Paradigm With Internet of Things: A Tutorial Introduction,” *IEEE Des. Test*, vol. 33, no. 2, pp. 76–96, Apr. 2016.

[5] M. Corici, B. Reichel, B. Bochow, and T. Magedanz, “An SDN-based solution for increasing flexibility and reliability of dedicated network environments,” in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016, pp. 1–6.

[6] M. Karakus and A. Durresi, “A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN),” *Comput. Networks*, vol. 112, pp. 279–293, 2017.

[7] M. Azab and J. A. B. Fortes, “Towards proactive SDN-controller attack and failure resilience,” in *2017 International Conference on Computing, Networking and Communications (ICNC)*, 2017, pp. 442–448.

[8] M. T. I. ul Huque, W. Si, G. Jourjon, and V. Gramoli, “Large-Scale Dynamic Controller Placement,” *IEEE Trans. Netw. Serv. Manag.*, vol. 14, no. 1, pp. 63–76, Mar. 2017.

[9] B. Kar, E. H. K. Wu, and Y. D. Lin, “The Budgeted Maximum Coverage Problem in Partially Deployed Software Defined Networks,” *IEEE Trans. Netw. Serv. Manag.*, vol. 13, no. 3, pp. 394–406, Sep. 2016.

[10] K. S. Mishra and A. K. Tripathi, “Some Issues, Challenges and Problems of Distributed Software System,” *Int. J. Comput. Sci. Inf. Technol. Varanasi, India*, vol. 7, p. 3, 2014.

[11] T. Koponen *et al.*, “Onix: A distributed control platform for large-scale production networks.,” in *OSDI*, 2010, vol. 10, pp. 1–6.

[12] F. J. Rodríguez, S. Fernandez, I. Sanz, M. Moranchel, and E. J. Bueno, “Distributed Approach for SmartGrids Reconfiguration Based on the OSPF Routing Protocol,” *IEEE Trans. Ind. Informatics*, vol. 12, no. 2, pp. 864–871, 2016.

[13] K. Phemius, M. Bouet, and J. Leguay, “DISCO: Distributed multi-domain SDN controllers,” in *2014 IEEE Network Operations and Management Symposium (NOMS)*, 2014, pp. 1–4.

[14] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to reliable and secure distributed programming*, 2nd ed. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2011.

[15] J. Aspnes *et al.*, “Eight Open Problems in Distributed Computing.,” *Bull. EATCS*, vol. 90, pp. 109–126, 2006.

[16] J. Doyle, “scaling,” in *OSPF and IS-IS: Choosing an IGP for Large-Scale Networks: Choosing an IGP for Large-Scale Networks*, Addison-Wesley Professional, 2005.

[17] A. Tootoonchian and Y. Ganjali, “Hyperflow: A distributed control plane for openflow,” in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, 2010, p. 3.

[18] F. Botelho, T. A. Ribeiro, P. Ferreira, F. M. V Ramos, and A. Bessani, “Design and Implementation of a Consistent Data Store for a Distributed SDN Control Plane,” in *Dependable Computing Conference (EDCC), 2016 12th European*, 2016, pp. 169–180.

[19] S. Hassas Yeganeh and Y. Ganjali, “Kandoo: a framework for efficient and scalable offloading of control applications,” in *Proceedings of the first workshop on Hot topics in software defined networks*, 2012, pp. 19–24.

[20] M. Moradi, Y. Lin, Z. M. Mao, S. Sen, and O. Spatscheck, “SoftBox: A Customizable, Low-Latency, and Scalable 5G Core Network Architecture,” *IEEE J. Sel. Areas Commun.*, vol. PP, no. 99, p. 1, 2018.

[21] H. Tahaei, R. B. Salleh, M. F. A. Razak, K. Ko, and N. B. Anuar, “Cost Effective Network Flow Measurement for Software Defined Networks: A Distributed Controller Scenario,” *IEEE Access*, vol. 6, pp. 5182–5198, 2018.

[22] M. A. Harrabi, M. Jeridi, N. Amri, M. R. Jerbi, A. Jhine, and H. Khamassi, “Implementing NFV routers and SDN controllers in MPLS architecture,” in *Information Technology and Computer Applications Congress (WCITCA), 2015 World Congress on*, 2015, pp. 1–6.

[23] M. Arregoces and M. Portolani, “Data center fundamentals,” in *understand data center network design and infrastructure architecture, including load balancing, SSL, and security*, Indianapolis: Cisco Press, 2003, p. 526.

[24] E. Alasadi and H. S. Al-Raweshidy, “SSED: Servers Under Software-Defined Network Architectures to Eliminate Discovery Messages,” *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 104–117, Feb. 2018.

[25] J. Moy, “RFC 2328: OSPF Version 2,” 1998.

[26] P. S. (Lucent Technologies) and M. H. (Lucent Technologies), "RFC2663:IP Network Address Translator (NAT) Terminology and Considerations," 1999.

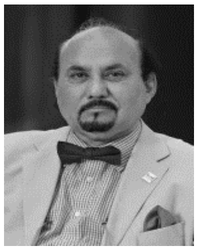
[27] Open network foundation, "Openflow switch specification, version 1.5.0," 2014.

[28] R. Kubo, T. Fujita, Y. Agawa, and H. Suzuki, "Ryu SDN framework-open-source SDN platform software," *NTT Tech. Rev.*, vol. 12, no. 8, 2014.

[29] O. V. Switch, "Open vSwitch," 2014.

**Emad Alasadi** received the B.Sc. degree in computer and software engineering from Al-Mustansiriyah University, Baghdad, Iraq, in 2003 and the M.Sc. degree in computer engineering and information technology from University of Technology, Baghdad, Iraq, in 2006. He is currently pursuing the Ph.D. degree in Electronic and Computer Engineering at Brunel University, London.

From 2007 to 2013, he was a lecturer with the university of Al-Qadisiyah, Qadisiyah Province, Iraq. His research interests include software-defined networking, network architecture, network functions virtualization, distributed system and language programming.



**Hamed Al-Raweshidy** (SM'03) is Professor of Communications Engineering and received his BEng and MSc from University of Technology, Baghdad in 1977 and 1980 respectively. He completed his Post Graduate Diploma from Glasgow University, Glasgow, UK in 1987. He received his PhD in 1991 from Strathclyde University in Glasgow, UK.
He currently is the Director of The Wireless Networks and Communications Centre (WNCC) at Brunel University London, UK. He has worked with The Space and Astronomy Research Centre (Iraq), PerkinElmer (USA), Carl Zeiss (Germany), British Telecom (UK), Oxford University, Manchester Met. University and Kent University. He has published over 380 papers in International Journals and referred conferences.