Journal of Computers Vol. 31 No. 5, 2020, pp. 73-88
doi:10.3966/199115992020103105006

DD-SDWMN: Dynamic Distributed Software Defined Wireless Mesh Network

Hisham Elzain, Wu Yang*

Information Security Research Center, Harbin Engineering University, Harbin, China
hisham_alzain21@yahoo.com, yangwu@hrbeu.edu.cn

Received 6 April 2019; Revised 31 August 2019; Accepted 13 October 2019

Abstract. Software-Defined Networking (SDN) firstly has emerged in wired networks, and then it has extended to wireless networks. Software-Defined Wireless Networking (SDWN) has a great potential to increase efficiency, ease the complexity of control and management, and accelerate technology innovation rate of wireless networks. To improve the reliability and performance of SDN's architectures, a logically centralized but physically distributed controller design has proposed. However, there are two key problems have appeared. First, the control plane network managing task. Second, the distributed controllers' deployment is manual and static. To address these two problems in Software Defined Wireless Mesh Networking (SDWMN), this paper presents a novel architecture called Dynamic Distributed SDWMN (DD-SDWMN) to bootstrapping SDN-based WMN architecture and dynamic deploying the distributed controllers. The controller needs to have updated reports of the network status' changes. The global controller aggregates entire topology discovery and monitor QoS properties of extended WMN nodes using LLDP protocol, which unable in multi-hop ordinary architectures. DD-SDWMN has implemented in Mininet-wifi emulator on top of POX controller and Advanced Message Queuing Protocol (AMQP). The results present the DD-SDWMN control plane consistency and two functional applications: topology discovery and QoS monitoring. The current results lead to future studies on QoS routing in WMN.

Keywords: QoS, SDN distributed control plane, SDWMN, SDWN, topology discovery, WMN

1 Introduction

Wireless mesh network (WMN) is a multi-hop network that regards as a wireless potential key technology with low-cost Internet coverage for specific areas. It has a great role in various scenarios of applications: public safety, transportation, enterprise networks, mining fields and emergency response, etc. WMNs deployment and use are a relatively quick and low-cost that is because of no need for any wired network as a backbone [1]. WMN consists of three types of nodes: mesh clients (any end-user wireless device e.g. Smartphone, laptop, etc), mesh routers (to build a wireless network backbone), and gateways (especial routers that have a capability to connect to the Internet). In infrastructure WMNs, mesh routers are static and can be equipped with various radio technology, such as WiFi (IEEE 802.11), ZigBee (IEEE 802.15.4) and WiMAX (IEEE 802.16) [2-3] towards a high coverage for the targeted area. But despite the steadily evolving of wireless technology standards, there are structural barriers still prevent wireless networks' infrastructure to openness and fulfill technical innovation requirements [4].

Software-Defined Wireless Networks (SDWN) [5-6] is the technical term of applying SDN's concepts to wireless networks. SDWM attempts to inherit (wired) SDN flexibility, it decouples radio controlling and radio data forwarding functions from each other. It supports wireless networks to become more programmable by abstracting the underlying wireless infrastructure from network services and applications through a higher-level Application Programmed Interfaces (APIs). So, wireless networks under SDN architecture will become more innovative ecosystems and break down mentioned structural barriers. But, due to wireless networks nature, they face application challenges such as SDN

* Corresponding Author

bootstrapping [7] (i.e. network control plane initialization), distributed controllers deployment and their key functionalities such as topology discovery, links quality specifications monitoring, among others [8]. In this work, we propose solutions to these challenges in SDN-based WMN which is termed as Software-Defined Wireless Mesh Networks (SDWMN) [9].

SDN can create resilient network architectures, as a physically centralized network control architecture (a single controller) or a physically distributed logically centralized architectures that rely on distributed controllers organized either hierarchically or flatly, to control multi-domains to achieve network reliability and scalability [10-11]. Despite the distributed control plane approaches lead to complexities in SDN controller development and management. However, these solutions are more receptive to network status changing and handling related events. Cause of controllers' distribution is closer to the network resource, than in the centralized architecture.

In the literature, there are several studies [12-14] distribute SDN controllers to handle the network control plane workload based on topology. They follow topological structures on controllers' distribution in network architecture in flat and hierarchical designs. They are applied within wired networks context, and in [12] work the controllers' deployment is still static, except [13] and [14] they adopted different dynamic approaches. In this research, we propose a new dynamic approach based on Advanced Message Queuing Protocol AMQP [15] to distribute controllers among WMN.

Any distributed architecture needs an orchestration framework to maintain all network domains topology [16]. So, an effective network topology discovery mechanism is essential for managing the network and deploying end-to-end applications and services on the top of the orchestration architecture among multiple distributed domains. The network global view is a critical factor in SDN architectures, and the controller provides such view through topology discovery service. Thus, the information offered by topology discovery is crucial for network applications such as routing, network resources allocation and management, and fault recovery that reside on the top of the SDN controller. In this context, topology discovery process time and load are fundamental for a timely and lightweight response. Therefore, up-to-date network topology discovery must use efficient mechanisms in SDN architectures, and it becomes one of the significant design metrics for SDN scalability [17]. Keeping an overall view of a large network generates a huge amount of information about the physical plane state. Furthermore, the massive volume of a control flow would produce a heavy load to the SDN controller which could degrade the controller performance [18], and creates problems in network scalability. Current SDN topology discovery schemes designed for wired networks, they do not suffice for wireless networks. Also, they do not collect any quality of Service (QoS) properties of underneath network elements.

This paper addresses distributed multiple-domain SDWMN as presented in Fig. 1, which can be deployed for various applications of WMN. Generally, they decomposed into geographical or administrative interconnected domains, each domain consists of wireless mesh routers that use one or more of mentioned various wireless standard technologies. For the experiment, we used IEEE802.11 standard with two frequency bands. One for data forwarding and the other for forwarding control messages. The work focuses on finding the suitable architecture of WMN, allowing use the state-of-the-art topology discovery and QoS monitoring based on Link Layer Discovery Protocol (LLDP). The (AODV) Ad-hoc On-Demand Distance Vector Routing is used in our recent application as one of the QoS routing protocols, and the packet delay is used as the main routing metric.

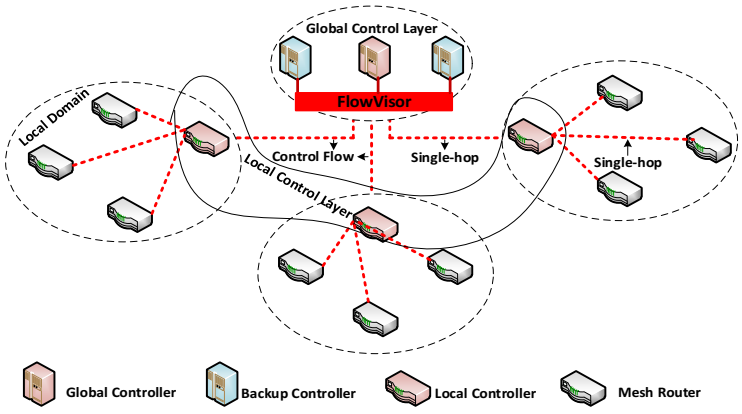


Fig. 1. DD-SDWMN architecture

The state-of-the-art distributed solutions of SDWN are not sufficient for WMN scalability; it needs a fine-grained control plane that depends on an efficient communication system for inter-controller exchange. We propose DD-SDWMN architecture, a Dynamic Distributed SDWMN. DD-SDWMN represents the heterogeneity and distributed nature that WMN calls for, to be more robust to failure and adaptable to user requirements. It distributes the control plane into two layers, global and local control layer. The Global Control Layer (GCL) consists of one main controller or Global Controller (GC) and backup controllers. The GC forms a global network view by gathering information from the Local Control Layer (LCL), which composes of a number of Local Controllers (LCs), each of which controls a local domain of mesh switches. The GCL-LCL control channel is dynamically plugged by agents that developed to aggregate network information in the GC from LCs to provide end-to-end services. We show how DD-SDWMN's control plane initializes dynamically to solve SDN bootstrap problem. After that, the architecture starts providing with its main functionalities such as network traffic engineering and disruption and attacks survival. Contrary to currently distributed SDN architectures, WMN based on DD-SDWMN is resilient enough to discriminate links with the best characteristic (bandwidth, latency, packet loss …) for data forwarding. We implement DD-SDWMN on top of POX [19] OpenFlow controller, and AMQP. For architecture performance evaluation, we present its functionalities on Mininet-wifi [20] emulator for SDWN. According to control plane consistency test and two use cases: topology discovery mechanism for multi-hop networks (the state-of-the-art topology discovery cannot apply for a multi-hop network such as WMN) and QoS monitoring.

The rest of the paper is organized as follows: Section 2 describes the DD-SDWMN architecture. It presents the modules and agents that compose the GC and LCs controllers. Section 3 discusses the proposed architecture implementation. The details of topology discovery and QoS monitoring are elaborated in Section 4, 5 and 6. The evaluation of the work through experimental results is shown in section 7. Section 8 concludes the paper and presents future work direction.

2 DD-SDWMN Architecture

In SDWMN, a centralized controller controls all the switches (mesh routers) of the network (controller-switch connection is in a multi-hop). Each switch in the network needs to rely on the central controller forwarding decision. Thus, for every new flow, each switch generates a request to the controller, which responses with appropriate flow entries messages. Obviously, this scenario costs the controller more load and decreases network performance. Contrary, in DD-SDWMN architecture the network control has logically divided via network slicing into two control layers (controller-switch connection is in a single-hop): the Global Control Layer (GCL) and the Local Control Layer (LCL). In this section, we present DD-SDWMN's architecture.

2.1 Overall Architecture

DD-SDWMN is a distributed multi-domain SDWMN control plane that enables end-to-end network services delivery efficiently. DD-SDWMN control plane composes of two types of SDN controllers: Local Controllers (LCs), each of which in charge of a local domain, and a Global Controller (GC) that communicates with LCs to aggregate network-wide information forming a global network view in order to end-to-end efficient management. Figure 1 shows the architecture, it presents that DD-SDWMN's control plane composes of two layers: the GCL and LCL. GCL maintains a global view of the entire network and LCL defines the local control of the distributed control domains. The main Global Controller (GC) and backup GCs controllers represent the Global Control Layer. The east-west interface is used to GCL synchronization. The main GC selects switches that have a control capability to act as Local Controllers (LCs) upon its authorization. After that, each LC builds its own domain. As a result, any new flow arriving at the domain's nodes does not need to request the GC. Instead, the corresponding LC directly handles the flow. The LCs controllers shape the Local Control Layer. In addition, southbound interfaces use to push forwarding policies to data plane elements and gather their status. Finally, the northbound interfaces provide controllers with applications' management policies and requirements.

A controller GC or LC is composed of multiple modules. All of them are managed (start, stop, update, and communicate) by the core component. DD-SDWMN architecture leverages from existing SDN

controllers' modules, such as OpenFlow driver for OpenFlow protocol implementation, switch and host managers for network elements keep tracking and link discovery that implements LLDP (Link Layer Discovery Protocol) for topology discovery. Furthermore, we have developed new modules to enhance the architecture functionality in GC and LCs level, as in Fig. 2 and Fig. 3 respectively. GC controller must communicate with LCs to receive and accumulate overall network status information. In order to accomplish this task, it uses two key parts: (1) Coordinator module which establishes and maintains a reliable and secure distributed publish/subscribe channel. And (2) different agents that exchange needed information with LCs via this channel. The rest of this section presents these modules and agents at both GC and LC level.

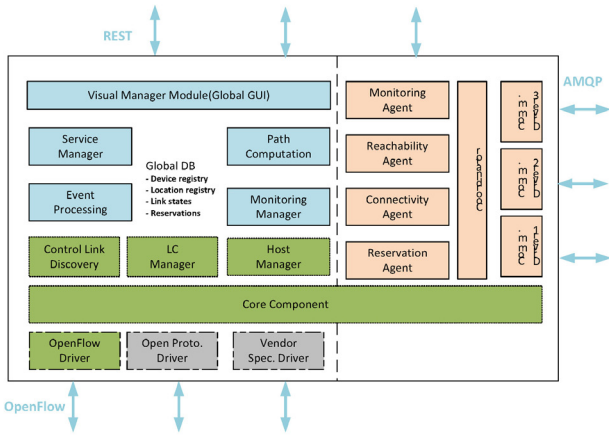


Fig. 2. Global Controller architecture

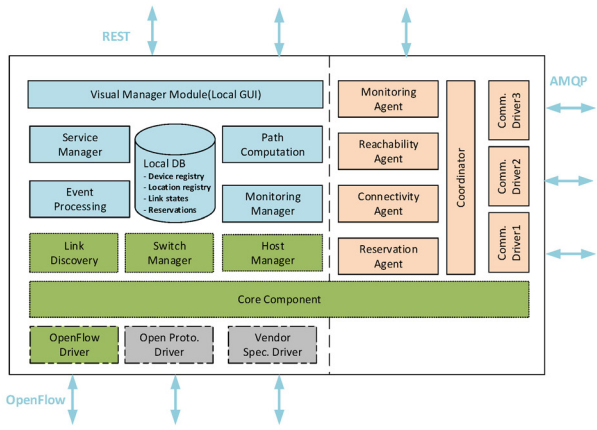


Fig. 3. Local Controller architecture

2.2 Modules Functionality

The DD-SDWMN architecture data plane composes of OpenFlow enabled mesh routers (switches). Some switches have a capability to act as local controllers, they seeking for GC controller by advertising themselves. Then the GC starts the authentication procedure to add them to the control plane. After that, each LC controller starts building its local domain, and triggers its control modules functionality such as topology discovery, QoS monitoring, and flow management enables them to react to network state (link is broken, high latency, bandwidth degradation …) dynamically by stopping and/or redirecting a traffic according to its criticality. This work is extending to our previous work [21].

The Local Data Base LDB. It is a database in which an LC stores its domain knowledge on topology, QoS monitoring and ongoing data flow. Other modules and agents use this information to reach the ultimate goals, i.e. taking suitable action on flows within a local domain. On the other hand, the Global Data Base GDB is a central database at the GC that accumulates entire network information from local domains' Local DBs.

The QoS Monitor module. It gathers QoS information such as bandwidth, delay, jitter, and packet loss of links between switches of a domain using customized LLDP packets, as in section 5. And all domains QoS information is aggregated at the corresponding module at GC.

The Local path computation module. The computing routes from source to destination using the Dijkstra algorithm taking into account QoS metrics of links in between. This module consults LDB for QoS monitoring information and network topology. The global path computation can compute the inter-domain path using data provided by local path computation modules.

The Service Manager module. It works at LC and GC to support end-to-end provisioning by receiving requests from northbound APIs for efficient network SLAs (Service Level Agreements) management at a local domain and entire network respectively. It verifies SLAs feasibility and respect for consulting other modules.

The virtual manager module. It resides on top of the other modules. It offers a (local\global) GUI through network virtualization to interact with other modules. It gathers information from many down modules and displays them to the network operator. Also, it provides him\her with network important parameters such as flow priorities, new routes, etc.

The Coordinator. This module establishes a control channel between the GC and LCs. It manages status information that frequently exchanges such as (link-state, host presence) and requests from LCs. hence, this communication channel should be reliable to guarantee the messaging process. For security and reliable delivery, AMQP is used for Coordinator implementation.

2.3 Agents Functionality

To collect and maintain a global network view at the GC level, in order to network efficient supporting for QoS routing and reservation functionalities, we define and implement the following agents.

The connectivity agent. It is in charge of maintaining peering channels between the GC and LCs. The work of this agent-based on event-driven fashion, it only sends information if a new LC is discovered or existing peering channel is changed. This information is grouped and maintained into the GDB, like information accumulated by the other agents. Eventually, the connectivity information used to take a global routing decision by path computation module.

The monitoring agent. GC periodically receives information (QoS properties) about available links of a domain from its LC, to inform about domain's capabilities, in order to support traffic transmission among network domains.

The Reachability agent. It advertises hosts presence on local domain margins so they become reachable. It offers a roadmap between domains' switches and hosts.

The reservation agent. It concerns on overall network flow setup, e.g. link teardown, and updates requests and application requirements such as QoS. It is like the Resource Reservation Protocol (RSVP). Each LC can locally handle these requests by the service manager. The reservation agent in the GC directly communicates with LCs on local domains along the path that needed to create or maintain.

In order to DD-SDWMN control plane consistency, the agents publish and consume messages that manage the required topics. The aggregated information that concern reachability (reachable hosts list in the network), connectivity (a list of LCs), and monitoring (peering transit paths status, in terms of QoS metrics...). This way, the GC is able to build a view of the entire network, thus it has the capabilities to perform path reservation, routing and manage SLAs.

3 DD-SDWMN Implementation

We have implemented DD-SDWMN architecture on top of the POX an open-source controller. There are some modules taken from POX's Python code with a little modification, other modules we have developed in Python, to manage GC-LCs communications. The agents located in GC and LCs notifies links states, devices location, and requests of path reservation using this control channel.

3.1 Coordinator Implementation

The coordinator is implemented as any other POX application. It receives Packet-IN messages from the Core module after subscription, sends its own Packet-OUT messages, reads and writes information from/to the GDB and at startup it views the POX controller configuration file. For coordinator operation, there are configurations parameters should determine: agents_list, messaging_server_type, and messaging_server_listening_port. Also, other optional parameters can be specified. An agent is a small class that handles GC-LCs exchanges from local managing modules to aggregated global modules and vice versa. Coordinator activates specific agent from the agents_list. It specifies the port that it reaches by the messaging_server_listening_port. From the messaging_server_type parameter, the coordinator determines the messaging driver that it uses, for current implementation we use RabbitMQ [22] driver that uses AMQP (federation mode), but also the DD-SDWMN architecture allows using other AMQP implementations such as Active MQ.

Coordinator implements an extended LLDP (Link Layer Discovery Protocol) version, which we call Coordinator-LLDP (C-LLDP) that uses in discovery functionality. C-LLDP message contains OpenFlow option added to the regular LLDP message. And IEEE has allocated the Organizationally Unique Identifier (OUI) to OpenFlow. Coordinator at GC sends these messages announcing the global controller existence, in order to discover mesh switches that have a controller to activate and add them as domains' local controllers (LCs). And at the LC level to discover and add switches to a local domain. At this end

coordinator stops sending the discovery messages and establishes the AMQP connection at the GC and LCs levels, otherwise, it continues sending periodically discovery messages at both levels. C-LLDP message as following contains required information to reach a controller (GC or LC).

```
"0x7F (127 – LLDP's Custom TLV type)
 0x00 0x26 0xE1 (OpenFlow OUI)
 0x17 (Messenger subtype)
 0x02 (controller ID)
 0x03 (switch ID)
 0x04 (switch port)
 0x05 (server IP)
 0x06 (server port)
 0x08 (server name)"
```

Coordinator offers a communication channel (publish/subscribe) for GC-LCs exchanges between agents. Coordinator basic operation is done via two special topics: GC_ID.*.* topic, GC_ID being the GC identifier. Via this topic, LCs can directly send messages to GC. For instance, this is used for a local domain topology discovery update. The other topic is general.*.*, which enables the GC to send messages to all LCs. For example, when GC wants to set or update LCs forwarding tables[1]. Coordinator communicates with AMQP implementation via drivers. An AMPQ driver should support a set of functions:

(1) Subscribe (topic) / unsubscribe (topic): to add and remove a topic to/from the topic list.

(2) Send (topic, message): send to a particular topic a specific message.

(3) Pair (LC_ID) and unPair (LC_ID): GC function to create a control channel to an LC and the opposite function is to remove this channel when an LC fails and cannot able to receive information from GC.

The coordinator also maintains LCs existence by sending periodic Keep-Alive messages. In the case that 3 contiguous Keep-Alive messages have no responses, the GC inferences an LC failure and triggers a mitigate procedure for this failure. The Coordinator application is flexible enough for an extension without altering the main classes of POX. Developers can provide a new driver for the implementation of AMQP to extend the Coordinator class, or by adding a new agent function.

3.2 Agents Implementation

Agents use Coordinator to exchange the information between the GC and distributed LCs. In this architecture, four agents have developed: Reachability, Connectivity, Monitoring, and Reservation (see section 2.2). They publish on particular topics, e.g. monitoring.ID.bandwidth.2s which the monitoring agent at the LC that identifier by ID advertises the remaining bandwidth it can offer for traffic transmission. Upon a global agent at the GC receives information from a corresponding local agent at an LC, it stores this information at the GDB. Then the global modules aggregate this information to form a global network view, which allows the GC to make efficient decisions on flows all over the network.

To end-to-end provisioning, the reservation agents implement a mechanism like RSVP reservation protocol. In which agents exchange reservation requests responding to flow descriptors. Coordinator and its assistants (drivers, agents …) development add extra lines of code and their operation consuming extra memory to the POX controller. The coming two sections show two cases to asses DD-SDWMN architecture implementation.

4 Topology Discovery

In order to efficient operation of SDN based network services and applications, they need to have updated information that describes network state, particularly the network topology. The controller is responsible for offering this information using an efficient and reliable method. Generally, no any dedicated functionality that OpenFlow devices support for topology discovery. However, most of the

[1] LC works as a forwarding device beside it is a local controller.

current SDN controller systems use the same approach implemented by the original SDN controller (i.e. NOX controller) [23]. This mechanism named OpenFlow Discovery Protocol (OFDP) [24], and it has used as a topology discovery de facto scheme. OFDP leverages the Link Layer Discovery Protocol (LLDP) [25], which allows switches in LAN (IEEE 802 Local Area Network) to advertise their capabilities with each other. Fig. 4 depicts a simple scenario for topology discovery, the process details are mentioned in [24]. Although OFDP mechanism has adopted by the majority of the current controller platforms, the controller sufferers from the number of messages load during the process. For every cycle of topology discovery, the controller sends Packet-Out messages per every active port of the switches and receives Packet-In messages twice the number of links between them. Since topology discovery is a periodical process, OFDP mechanism affects controller performance, because the number of packets in/out to/from a controller is depending on the number of network switches and their active ports. For that, the heavy load of the topology discovery process in a controller is increased according to network scale. And due to the operation mechanism of LLDP protocol, it cannot able to discover multi-hop links between switches in pure or hybrid OpenFlow networks. Therefore, there are two main hitches of the current topology discovery approach. One is the controller overload and the other is this approach works only in the single-hop network. To solve the first problem i.e. controller overload, Pakzad et al. [24] proposed a new version of OFDP, they called it OpenFlow Discovery Protocol version 2 (OFDPv2). It reduces the number of Packet-Out messages to one message per a switch, instead of one per each active port of a switch. The work proofed that the modified version is identical in discovery functionality to the original version. Furthermore, it achieves the aim with a noticeable reduction in a number of messages exchange and reduces the discovery induced CPU load of the controller around 45% comparing to OFDP.

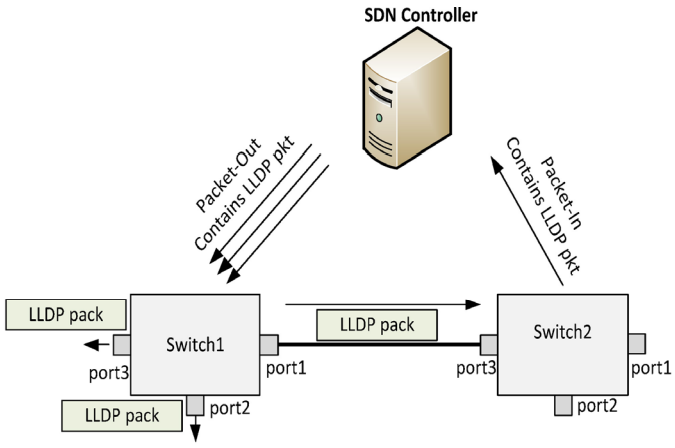


Fig. 4. OFDP topology discovery simple scenario

To solve the second problem, i.e. the disability of applying OFDP or even OFDPv2 on multi-hop networks, that is because both OFDP and OFDPv2 leverage LLDP packets as mentioned above, and LLDP packets addressed as a "bridge-filtered multicast address", so they are a single hop and not forwarded across switches [24]. In hybrid networks where there are one or more traditional switches (do not support OpenFlow) between OpenFlow switches, they process the LLDP packets and drop them. Therefore, the controller should use a combination of LLDP and BDDP (Broadcast Domain Discovery Protocol) to discover indirect links between OF switches ports in the same broadcast domain [26]. However, the work does not present any solution for pure OF switches multi-hop networks. Adapting OFDP Topology discovery approach to SDN based wireless multi-hop networks is a challenge. It cannot collect wireless nodes and links characteristics, such as node related attributes: localization and QoS properties and links nature: not point-to-point and fixed capabilities connection like wired links, Chen et al. [27] proposed which called "A Generic and Configurable Topology Discovery" to analyze the general topology discovery representation required by SDN applications and how the SDN controllers offer it.

DD-SDWMN architecture considers as a direct work that adapts the state-of-the-art topology discovery in wireless multi-hop networks.

4.1 Aggregated Topology Discovery Mechanism

Among the DD-SDWMN architecture, the aggregated topology discovery mechanism can be able to apply the topology discovery protocol (OFDP) in multi-hop networks such as wireless mesh network. The mechanism is based on OFDPv2, and benefited from the breaking of the multi-hop control channel only into two single-hops by dividing the process into two phases: local domain topology discovery and global network topology discovery, the proposed mechanism reduces the heavy load of the topology discovery on the GC controller.

4.2 A Local Domain Topology Discovery

After completing their local domains construction, the LCs controllers start topology discovery service (Link discovery module) to discover local domains' nodes. Precisely, in this process, the LC controller concerns on link discovery, and it does not need to rediscover the domain nodes (switches) since they are already have initiated a connection to the controller. The controller sends an individual Packet-Out each of which contains an LLDP packet, with a rule to send the specific packet out on the corresponding interface. Then via Packet-In message, the LLDP packets send to the controller obeying to the pre-installed rule that says, "Forward any received LLDP packet from any interface except CONTROLLER interface to the controller". The process is repeated per every switch in the domain, to discover active links between them. The entire domain topology discovery process is periodically performed per every 5 seconds as default interval size of the NOX controller. Ultimately, LCs maintains up-to-date local domains topology information in their LDBs, and the connectivity agent is ready to inform GC about domain topology.

4.3 Global Network Topology Discovery

Since the GC controller knows all the LC controllers during DD-SDWMN architecture initialization. Firstly, the GC controller - particularly Control link discovery module - discovers the links between LCs as the same scenario mentioned in 4.1.2, because of all the LCs in a single-hop from the GC controller. Secondly, via AMQP the global connectivity agent requests each LC to send its local domain topology information, to aggregates topology information for the entire network in the GDB. By using this hierarchically aggregated mechanism over DD-SDWMN architecture, the heavy topology discovery computation burden on the GC can reduce and the slow convergence time problem of the distributed nodes of high scale WMN can be addressed. Fig. 5 depicts the aggregated topology discovery mechanism control messages flow between the GC and the LCs.

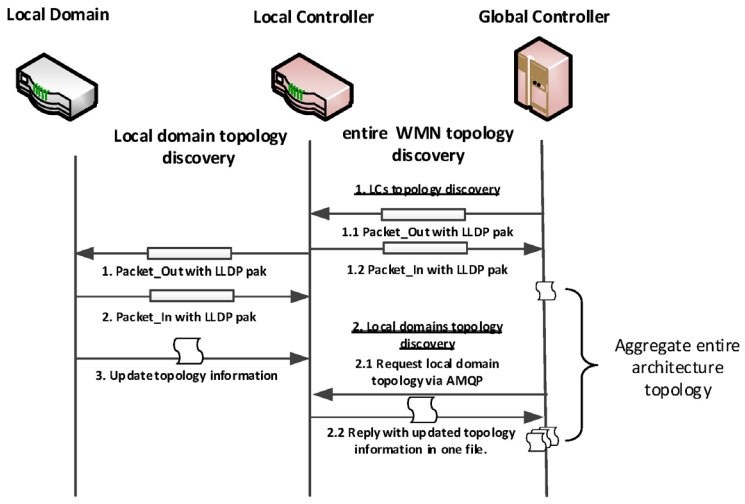


Fig. 5. Messages flow for the aggregated topology discovery mechanism

5 QoS Monitoring

Quality of service (QoS) of Network forwarding devices is crucial particularly for real-time applications like video streaming. Current SDN topology discovery service does not monitor or collect QoS of underline network elements. However, it is based on the LLDP protocol, which has the ability to collect QoS properties [28]. As shown in Fig. 6(a), additional to mandatory TLVs fields that used in topology discovery, LLDP has optional Type Length fields that can be customized to discover other features such as QoS properties. In [28] four optional fields identified to carry bandwidth, delay, jitter, and packet loss, with 8 bytes size of each property as shown in Fig. 6(b). Therefore, customized LLDP packet for QoS collection is longer than original LLDP packet by 38 bytes. Since the aggregated mechanism allowed applying LLDP-based topology discovery on DD-SDWMN, it is also can monitor and collect QoS.

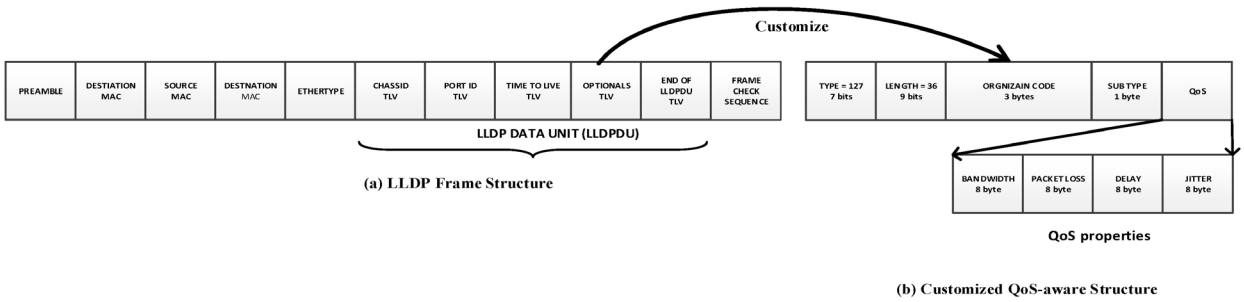


Fig. 6. LLDP Packet Format for QoS monitoring

6 The Controller Traffic Overhead

Controller load and performance is a critical factor of SDN scalability. Since the SDN controller (GC or LC) runs topology discovery service continuously, therefore, it is important to calculate and know the load that this service exploits the controller. As we mentioned above, the DD-SDWMN architecture offers the appropriate environment for applying topology discovery based on LDDP. Due to the current topology discovery mechanism, the controller load is depending on the number of Packet-Out messages that the controller should send addition to the number of receiving Packet-In messages. Discovering a local domain topology is not different from the state-of-the-art mechanism. In every discovery cycle, the amount of received LDDP Packet-In (P_{IN}) messages by the LC controller depends on a number of domain nodes. Actually, it is twice the amount of active inter-switch links within the domain, a packet per each link direction. On the other hand, the total amount of LLDP Packet-Out (P_{OUT}) the message that sends by the LC controller per every cycle is equal to the total number of switches.

With N being the number of domain's switches, L the number of inter-switches links, and f_i the number of interfaces of a switch, same as in a single controller single-hop architecture, the number of messages in/out the controller can express as follows:

For current mechanism (OFDPv2):

$$P_{IN} = 2L. \tag{1}$$

$$P_{OUT} = N. \tag{2}$$

A significant reduction of the LLDP Packet-In and Packet-Out message numbers can be achieved in DD-SDWMN architecture when discovering entire network topology using the aggregated topology discovery mechanism. The GC controller load calculated in two stages: firstly, when discovering links between LCs and it depends on the number of LCs. Secondly, the load of aggregating local domains topology information from each LC as a single message for each domain using connectivity agents of AMQP.

With L_{LC} being the number of LC controllers, L_{LC} the number of inter-LCs links, and f_i the number of the interface of LCi can express as follow:

$$P_{IN} = 2L_{LC} + N_{LC} \,.\, (L_{LC} < \text{the total of entire network link}) \quad \textbf{(3)}$$

$$P_{OUT} = L_{LC} \,.\, (N_{LC} < \text{the total of entire network nodes}) \quad \textbf{(4)}$$

Together with the DD-SDWMN architecture, a suitable environment has been existed to apply current topology discovery in WMN. Moreover, it reduces the load of the GC controller. It reduces the number of the direct topology discovery i.e. Packet-In message to one per domain and Packet-Out to be equal to the number of LCs regardless of the number of domain nodes, and after AMQP starts the connectivity agents send topological information messages on-demand. Then the load calculation of direct discovery messages in the GC depends only on the number of LCs instead of entire network switches.

On the other hand, customizing and extending LLDP packet for QoS monitoring despite it extends LLDP packet size, but the traffic flow is slightly different as in topology discovery service, as in7.3.

7 Evaluation

This section presents how we assessed DD-SDWMN capabilities. Addition to main functions to this architecture such as QoS routing and reservation, it aims to be resilient to disruptions in the control plane (controller failure, GC-LC communication failure) or in the data plane (inter-switches link failure). Thus we present a control plane adaptation test and two use cases to evaluate DD-SDWMN features.

7.1 Experimental Setup

To verify the feasibility of our proposed model, we conducted experiments using Mininet-wifi the Software-Defined Wireless Network emulator, it creates a network of virtual SDN switches (routers), stations (hosts), and wireless channels (links). We exactly used Open vSwitch [29], which is a software-based virtual SDN switch with OpenFlow support, to create mesh routers. Mininet-wifi was run in VirtualBox as virtualization software, and Linux used as the hosting operating system. As we mentioned previously, the POX controller used as our experimental SDN controller platform and programming coding to implement proposed architecture have implemented in python. All the software used for experiment prototype implementation summarizes in Table 1.

Table 1. Software used in Experiment

Software	PURPOSE	Version
Ubuntu Linux	Hosting OS	16.04
Mininet-wifi	SDWN Emulator	2.2.1d1
POX	SDN Controller	Dart branch
FlowVisor [30]	Network slicing	1.2.0
Open vSwitch	SDN Virtual Switch	2.0.2
Python	Programming Language	2.7

All experiments were performed on a Dell Laptop (Inspiron15 5000 Series) with an i7-8550U Intel CPU, running at 1.8GHz, with 12GB DDR4 2400MHz RAM.

7.2 Control Plane Adaptation

Here we show how DD-SDWMN's control plane exchanges control information and self-adapt to the network state. To mitigate any congestion in control-messages exchange link between GC and LCs, agents offload traffic of a weak direct control connection (via a dedicated control channel (out-of-band)), by identifying alternative routes among data plane (in-band) or reducing the control messages frequency on it if no any alternative route. For instance, the monitoring agents usually send information every 3s, and this period changes to 10s in weak links. Also, connectivity and reachability agents can send their information via alternative routes. But, they are event-driven agents contrary to the monitoring agent.

Upon control plane bootstrapping and LCs discovery, LC1 and LC2 are connected to GC as depicted in Fig. 7. In this scenario, the control-link between GC and LC1 is congested (its latency => 50ms). Thus the monitoring agent relaying control messages between GC and LC1 via the control link of LC2, and

through the data link between LC2 and LC1 to offload that congested control link.

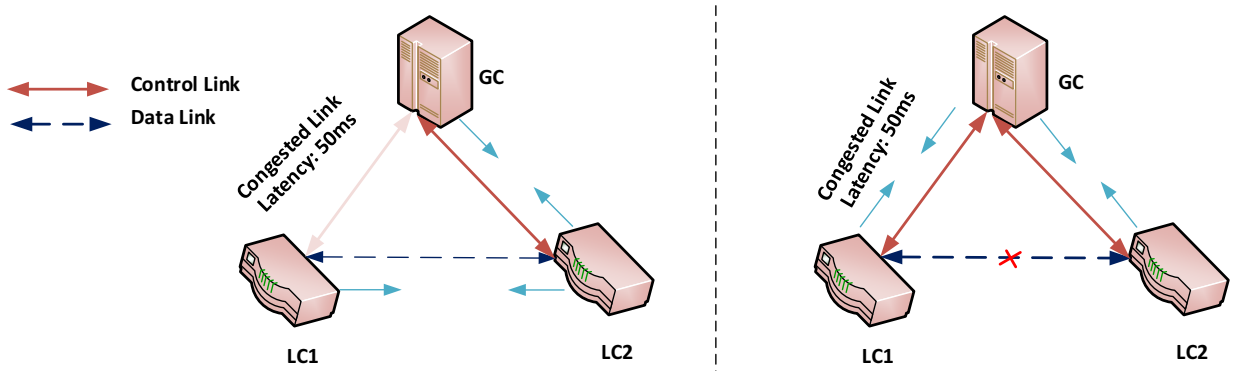
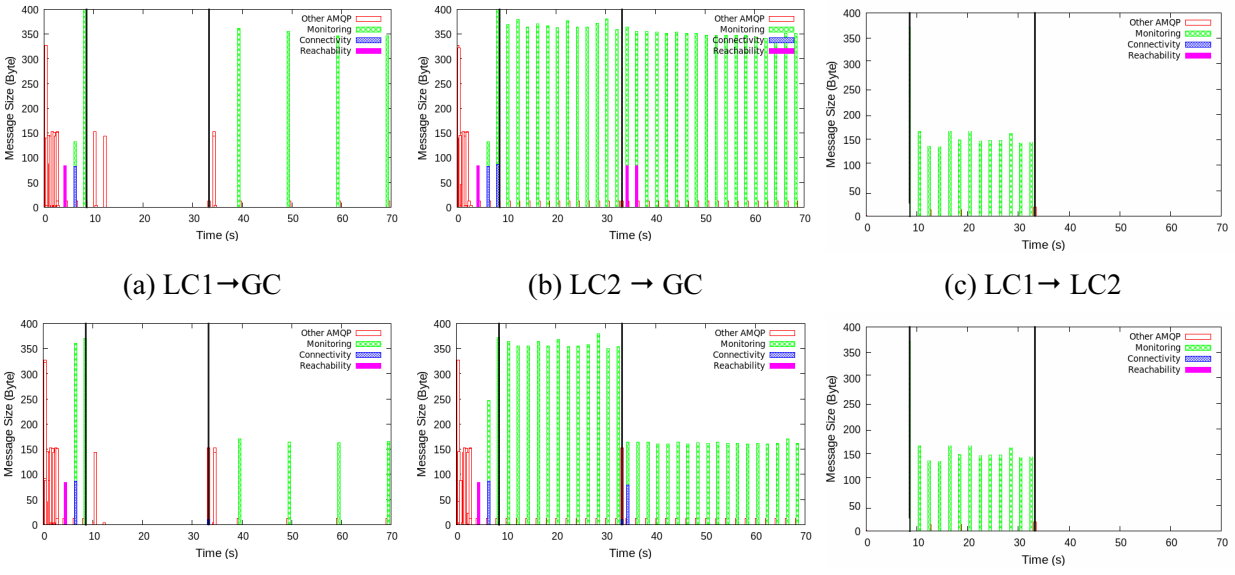


Fig. 7. control-plane adaptation: (Left) the congested situation: GC and LC1 use data forwarding link between LC1 and LC2 and via LC2 to GC. (Right) Data link disruption: GC and LC1 use the congested control link with adapting information exchange frequency, when LC1↔LC2 has broken down

Fig. 8 presents the conducted evaluation to show DD-SDWMN control plane adaptation with network conditions, e.g. LC1↔LC2 data link failure. The figure shows the utilization of the link in both directions, left after GC discovers LC1 and LCs and AMQP system starts messages exchanging between them. The TCP payload of received packets split into three stages:

- Local controllers discovery: it took the first 9 seconds where LCs exchanging their capabilities to advertise themselves. AMPQ communication system was active during this period because the brokers should establish the control channel and subscribe to available topics. During this stage, the monitoring process already has started but not adapted to the weak link yet.
- Monitoring adaptation: after the end of the previous stage till t = 33s, in this period the weak link was discovered by the monitoring agents and they started adaptation behavior. As observe on Fig. 8(a) and Fig. 8(d) the monitoring was stopped from t = 10s because the GC↔LC1was congested (weak) at the time, as in Fig. 8(b) and Fig. 8(e), the monitoring traffic increased over the GC↔LC2 (control link), and appeared LC1↔LC2 (data link) as in Fig. 8(c) and Fig. 8(f).
- Failure recovery: started right after the alternative link (LC1↔LC2) is cut at t = 33. Thus the monitoring information is forwarded via the congested link (GC↔LC1) but in adapted frequency, as in Fig. 8(a) and Fig. 8(d).



(a) LC1→GC (b) LC2 → GC (c) LC1→ LC2

(d) GC → LC1 (e) GC → LC2 (f) LC2 → LC1

Fig. 8. monitoring information exchanges adaptation. Different agent messages' packets, the bootstrap & discovery stage at t = 9s. The link was cut off at t = 33s, LC1↔LC2 data link is cut off

7.3 Aggregated Topology Discovery

We implemented the aggregated topology discovery mechanism after performing the required modification on the POX Link discovery module (discovery.py) in python, to apply the enhanced version of OFDP (OFDPv2). Extensive tests performed on the architecture to establish OFDPv2 functionality. As expected, it implemented in WMN as one of the multi-hop networks. The main goal of our evaluation is to prove that using the DD-SDWMN architecture lets it is possible to use the same approach of the current SDN topology discovery, as in section (4.1.2), no difference between the current mechanism and the first phase of the aggregated mechanism in regards to the local controller's topology discovery. Our focus is on the global controller load that produces by LLDP packets that the controller sends and receives.

The advantage of the aggregated mechanism is the number of packet-In messages is reduced to be only twice the number of active inter-local controller links (L_{LC}) addition to the number of messages (LCs) that represent the topology information of the local domains as in equation (3). Also, the total number of Packet-Out messages reduced to only one per domain or local controller (the number of domain members depends on the network operator setting), as shown in equation (4).

Since the key parameters that impact the controller load in OFDP are the number of switches and their ports [24], for that we can compare aggregated mechanism with OFDPv2 to see how much it has improved topology discovery load on the global controller. We calculate *G* the gained efficiency in terms of reduction of the number of direct packet-In and packet-Out messages, for mesh network with *N* switches, N_{LC} local controller, and f_i interface for a (switch/local controller) *i* as follow:

$$G_{Packet-IN} = \frac{P_{IN-OFDPv2} - P_{IN-aggregated}}{P_{IN-OFDPv2}} = \frac{2L - 2L_{LC}}{2L} = 1 - \frac{2L_{LC}}{2L}. \quad \textbf{(5)}$$

$$G_{Packet-OUT} = \frac{P_{OUT-OFDPv2} - P_{OUT-aggregated}}{P_{OUT-OFDPv2}} = \frac{N - N_{LC}}{N} = 1 - \frac{N_{LC}}{N}. \quad \textbf{(6)}$$

We notice that the reduction gained will be greater for WMN with a big number of total routers. This will verify among experiments using Packet-Out because its calculation only depends on the number of mesh router and local controllers.

7.3 Discovery Control Messages

We created a wireless mesh network topology of 105 routers (*N* = 105), five of them selected to act as local controllers ($N_{LC} = 5$) by running a controller in each switch. Each local controller controls a local domain of 20 switches. We observed and collected the statistics of the GC controller outflow (Packet-Out) per topology discovery cycle. The experiment was run 10 times with similar outcomes, as expected. Table 2 presents measured (calculated) compared to observed (via Wireshark) results, they show the reduction of Packet-Out control messages, as well as the calculated efficiency gain *G* of aggregated over OFDPv2 as in equation (6). As we see the experimental results corresponding to equations (2) and (4) and the parameters of the topology.

Table 2. number of Packet-Out control messages

	Number of Packet-Out	Efficiency gain G
OFDPv2	105	95%
Aggregated	5	

Since the topology has 105 routers, five of them act as LCs. Regard GC controller OFDPv2 requires 105 packet-Out messages compared to 5 messages for the aggregated mechanism, as expected. It proofs that aggregated mechanism achieves so great reduction in the global controller required direct LLDP Packet-Out control messages, with up to 95% less than in OFDPv2.

As we mentioned before, the load of the controller is a critical and important factor of any SDN scalability and performance system. And as the reduction of packet-Out messages results in a direct enhancement of controller CPU performance. Then the aggregated mechanism which achieves a reduction in the number of both Packet-In and Packet-Out messages -instead of the only reduction of Packet-In in OFDPv2- will positively impact the GC's CPU load. However, such a reduction of the CPU load of the core of DD-SDWMN architecture (the global controller GC) is a great improvement compared to the state-of-the-art. There are additional benefits of aggregated mechanism we mention some of them here without evaluation. Such as the traffic reduction on a controller-switch channel by reduction of the flow of Packet-In and Packet-Out messages, particularly in SDN architectures that adopt an in-band controller.

7.4 QoS Monitoring Scenario

Here we test the ability of QoS monitoring. Fig. 10 depicts a simple scenario application of a video streaming over DD-SDWMN architecture, where H1 is the server that streams a video (that size is 400 MB and length is 10 min) to the client H2. Initially, QoS properties of switches' ports set as following: 5 Mbit for bandwidth and 2,500 μs for the delay. During the video streaming, QoS properties (bandwidth, delay, jitter, and packet loss) are changing due to the consumption of resources. The properties cumulated by monitoring agent at the GC controller prove that aggregated mechanism can also effectively monitor QoS changing over LLDP. Fig. 11 shows fluctuations curves (QoS over LLDP frontend GUI snapshots) of the QoS properties changes. The subfigures (a), (b), (c), and (d) present the real-time statistics of QoS properties bandwidth, delay, jitter, and packet loss, respectively, of interface S1-eth2 in the local domain1 (i.e., the second Ethernet interface eth2 of switch s1) collected by monitoring of the LC, and the monitoring process going on for all switches' interfaces of the domain. 15 seconds duration used as the default interval of LLDP.

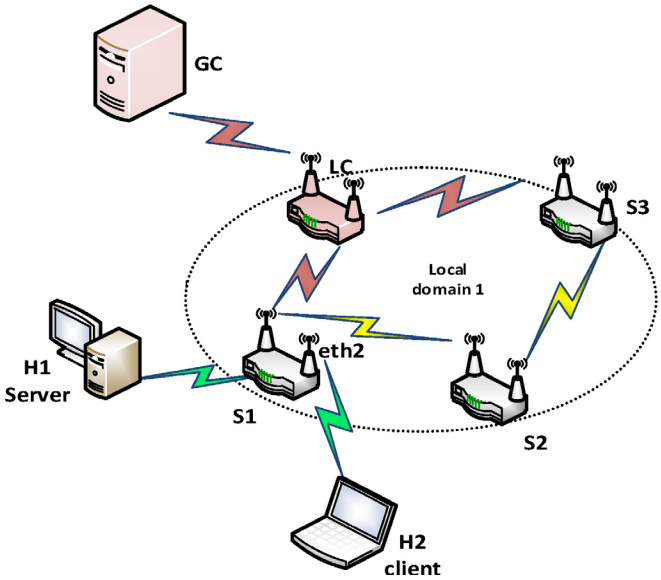
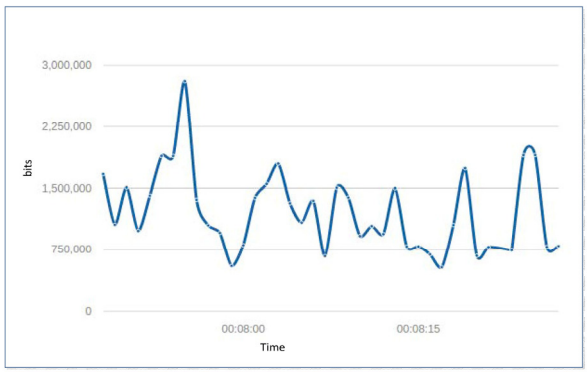
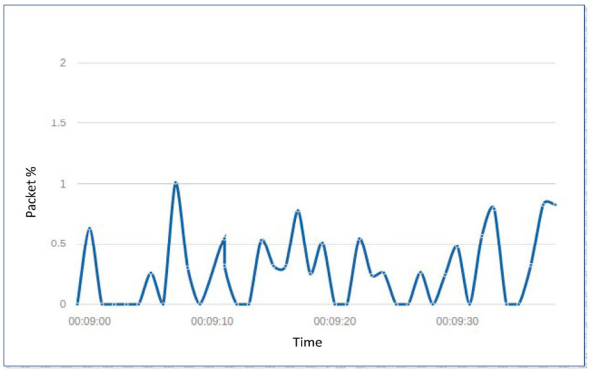


Fig. 9. A simple video streaming scenario

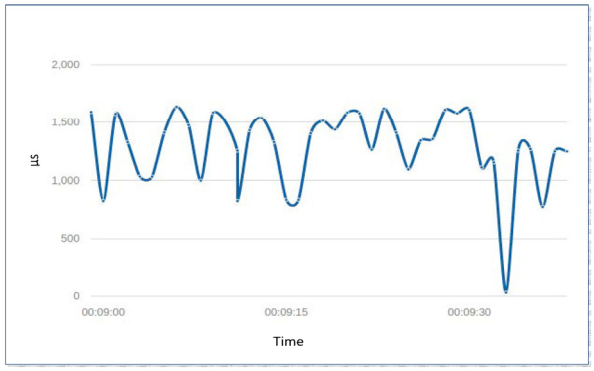
(a) S1-eth2: Bandwidth



(b) S1-eth2: Delay



(c) S1-eth2: Jitter



(d) S1-eth2: Packet loss

Fig. 10. QoS monitoring over LLDP Frontend GUI

Despite QoS monitoring over LLDP adds extra bytes when customizing the optional TLVs of LLDP packets for QoS properties collecting, no significant difference in network traffic compared to that caused by original packets, such as used in topology discovery. To evaluate this increasing of network traffic, we compare between pure LLDP packets (used in topology discovery) and customized LLDP packets (used in QoS monitoring). Both scenarios run separately during video streaming among the topology stated in Fig. 10. Network traffic captured using Wireshark, to measure the total network traffic and only LLDP packets (packet filter set to "LLDP") during 15 minutes. The evaluation results show slightly different in network traffic caused by QoS monitoring. The results show that the percentage of QoS monitoring is about .81% compared to .79% for topology discovery from the total packets flow. This proves that QoS monitoring over LLDP does not cause network traffic performance deterioration. Table 3 shows the evolution results.

Table 3. (QoS monitoring vs. topology discovery) over LLDP

LLDP packets for	network traffic			
	Total Packets	LLDP packets	Total bytes	LLDP bytes
QoS monitoring	459315	3725(0.81%)	2303289200	479700(0.021%)
Topology discovery	459250	3650(.79%)	2319671750	401500(0.017%)

8 Conclusion

In this work, we have proposed DD-SDWMN, a Dynamic Distributed Software-Defined Wireless Mesh Network multi-domain architecture. A main global controller (GC) distributes mesh routers in local domains and each domain is controlled by an elected local controller (LC). The distribution of the LCs

surrounding a huge number of mesh devices that cover large geographical areas supports WMN network extension. SDN controllers become closer to network edges for collecting and monitoring network status. Then, the GC aggregates network state information and forms the network global view. The GC-LCs communication achieves through the lightweight highly manageable control system, which consists of agents shared between GC and LCs. Agents dynamically plugged to aggregate network-wide information at the GC for end-to-end network services provisioning. We demonstrated the working of DD-SDWMN which responds resiliently and survives with network disruption. DD-SDWMN is a suitable architecture for multi-hop networks such as WMN in order to benefit from SDN controller services which are just applicable for single-hop networks e.g. Topology discovery service and QoS monitoring. We have implemented DD-SDWMN architecture on top of the POX OpenFlow controller and the AMQP protocol. The architecture functionalities evaluated according to control plane adaptation test and two use cases: aggregated topology discovery and QoS monitoring mechanisms. As future work, we decided to enrich DD-SDWMN architecture with additional self-healing and resilient recovery mechanisms to be more reliable to support QoS routing for video streaming over WMN.

Acknowledgements

This work is supported and funded by the International Exchange Program of Harbin Engineering University, for Innovation-oriented Talents Cultivation.

References

[1] F. Pakzad, Towards software defined wireless mesh networks, [dissertation], 2017.

[2] I.F. Akyildiz, X. Wang, W. Wang, Wireless mesh networks: a survey, Computer Networks 47(4) (2005) 445-487.

[3] K.N. Ramachandran, E.M. Belding-Royer, K.C. Almeroth, M.M. Buddhikot, Interference-Aware Channel Assignment in Multi-Radio Wireless Mesh Networks, in: Proc. 25th IEEE International Conference on Computer Communications, 2006.

[4] K.-K. Yap, R. Sherwood, M. Kobayashi, T.-Y. Huang, M. Chan, N. Handigol, N. McKeown, G. Parulkar, Blueprint for introducing innovation into wireless mobile networks, in: Proc. 2010 of the Second ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures, 2010.

[5] N.A. Jagadeesan, B. Krishnamachari, Software-defined networking paradigms in wireless networks: a survey, ACM Comput 47(2)(2014) 27-1.

[6] S. Costanzo, L. Galluccio, G. Morabito, S. Palazzo, Software defined wireless networks: unbridling SDNs, in: Proc. 2012 Software Defined Networking (EWSDN), 2012.

[7] C. Niephaus, G. Ghinea, O.G. Aliu, S. Hadzic, M. Kretschmer, SDN in the wireless context-towards full programmability of wireless network elements, in: Proc. 2015 of the 1st IEEE Conference on Network Softwarization (NetSoft), 2015.

[8] C. Chaudet, Y. Haddad, Wireless software defined networks: challenges and opportunities, in: Proc. 2013 Microwaves, Communications, Antennas and Electronics Systems (COMCAS), 2013.

[9] A. Abujoda, D. Dietrich, P. Papadimitriou, A. Sathiaseelan, Software-defined wireless mesh networks for internet access sharing, Computer Networks 93(2015) 359-372.

[10] F. Bannour, S. Souihi, A. Mellouk, Distributed SDN control: survey, taxonomy, and challenges, IEEE Communications Surveys & Tutorials 20(1)(2018) 333-354.

[11] M. Karakus, A. Durresi, A survey: control plane scalability issues and approaches in Software-Defined Networking (SDN), Computer Networks 112(2017) 279-293.

[12] M.F. Bari, A.R. Roy, S.R. Chowdhury, Q. Zhang, M.F. Zhani, R. Ahmed, R. Boutaba, Dynamic controller provisioning in software defined networks, in: Proc. 2013 Network and Service Management (CNSM), 2013.

[13] K. Phemius, M. Bouet, J. Leguay, Disco: distributed multi-domain sdnSDN controllers, in: Proc. 2014 Network Operations and Management Symposium (NOMS), 2014.

[14] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, G. Parulkar, ONOS: towards an open, distributed SDN OS, in Proc. 2014 of the Third Workshop on Hot Topics in Software Defined Networking. 2014.

[15] AMQP. <http://www.amqp.org>, 2013 (accessed 19.05.06).

[16] Open Networking Forum, SDN Architecture Overview, Version 1.1, document TR-504, Nov. 2014. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN-ARCH-Overview-1.1-11112014.02.pdf>, 2010 (accessed 19.06.02).

[17] L. Ochoa Aday, C. Cervelló Pastor, A. Fernández Fernández, Discovering the network topology: an efficient approach for SDN, Advances in Distributed Computing and Artificial Intelligence Journal 5(2)(2016) 101-108.

[18] M. Aslan, A. Matrawy, On the impact of network state collection on the performance of SDN applications, IEEE Communications Letters 20(1)(2016) 5-8.

[19] S. Kaur, J. Singh, N.S. Ghumman, Network programmability using POX controller, in: Proc. 2014 ICCCS International Conference on Communication, Computing & Systems, 2014.

[20] R.R. Fontes, S. Afzal, S.H. Brito, M.A. Santos, C.E. Rothenberg, Mininet-WiFi: Emulating software-defined wireless networks, in: Proc. 2015 11th International Conference on Network and Service Management (CNSM), 2015.

[21] H. Elzain, W. Yang, Decentralizing software-defined wireless mesh networking (D-SDWMN) control plane, in: Proc. 2018 of the World Congress on Engineering, 2018.

[22] RabbitMQ. <https://www.cloudamqp.com/blog/2015-05-18-part1-rabbitmq-for-beginners-what-is-rabbitmq.html>, 2013 (accessed 19.05.04).

[23] NOX, the original OpenFlow controller. <https://thenewstack.io/sdn-series-part-iii-nox-the-original-openflow-controller/>, 2013 (accessed 19.06.04).

[24] F. Pakzad, M. Portmann, W.L. Tan, J. Indulska, Efficient topology discovery in software defined networks, in: Proc. 2014 8th International Conference on Signal Processing and Communication Systems (ICSPCS), 2014.

[25] V.Z. Attar, P. Chandwadkar, Network discovery protocol LLDP and LLDP-med, International Journal of Computer Applications 1(9)(2010) 93-97.

[26] L.O. Aday, C.C. Pastor, A.F. Fernández, Current trends of topology discovery in OpenFlow-based software defined networks. <http://hdl.handle.net/2117/77672>, 2015 (accessed 19.06.04) .

[27] L. Chen, S. Abdellatif, P. Berthou, K.B. Nougnanke, T. Gayraud, A generic and configurable topology discovery service for software defined wireless multi-hop network, in: Proc. 2017 of the 15th ACM International Symposium on Mobility Management and Wireless Access, 2017.

[28] X. Chen, J. Wu, T. Wu, The Top-K QoS-aware Paths Discovery for Source Routing in SDN, KSII Transactions on Internet & Information Systems 12(6)(2018) 2534-2553 .

[29] Open vSwitch. <http://openvswitch.org>, 2011 (accessed 19.06.03).

[30] Flowvisor wiki. <https://github.com/OPENNETWORKINGLAB/flowvisor/wiki>, 2013 (accessed 19.06.04).