Automatic Configuration of OpenFlow in Wireless Mobile Ad hoc Networks

Sachin Sharma[1], Avishek Nag[2], Paul Stynes[3], and Maziar Nekovee[4]
National College of Ireland[1,3], University College Dublin[2], and University of Sussex[4]
Email: sachin.sharma@ncirl.ie[1], avishek.nag@ucd.ie[2], pstynes@ncirl.ie[3], M.Nekovee@sussex.ac.uk[4]

***Abstract*—A Mobile wireless Ad hoc NETwork (MANET) is a decentralized wireless network in which mobile wireless nodes either directly communicate with each other or communicate via other wireless nodes in the network. In addition, OpenFlow has disruptive potential in designing a flexible programmable network which can foster innovation, reduce complexity and deliver right economics. In recent years, there are significant interests from research communities to deploy OpenFlow in MANETs. This paper proposes a configuration method with which OpenFlow can be deployed automatically in a MANET without any manual configuration. The proposed configuration method is tested in an emulated MANET created on the Fed4FIRE testbed using Mininet-WiFi (an emulator for wireless software-defined wireless networks). Experimentation includes automatic configuration of OpenFlow in linear, sparse, and dense mobile ad hoc networks. Results show the effectiveness of the method in configuring OpenFlow in wireless mobile ad hoc networks.**

***Index Terms*—MANET, Software Defined Networking (SDN), OpenFlow, OLSR, Open vSwitch**

I. Introduction

A Wireless Mobile Ad hoc NETwork (MANET) is a collection of mobile wireless nodes that dynamically form a temporary network for communication without the use of a fixed infrastructure (e.g., access points or base stations in infrastructure wireless networks). MANETs are suitable for situations where an infrastructure network is not available or not trusted or costly to set up. Applications of MANETs include network connectivity in remote areas or in disaster scenarios such as earthquakes and tunnel accidents. Recent applications of MANETs include WiFi mesh networks in the context of Industry 4.0 and factory automation applications [1]. In addition, the integration of cloud and MANETs provides the facilities to access the cloud inside a MANET of smart devices where one or more smart devices may not have access to the Internet [2], [3].

Software Defined Networking (SDN) is an approach that facilitates network management by enabling programmatic efficient network configuration. OpenFlow [4] is the de facto SDN protocol for communication between the control and data plane of network devices. OpenFlow allows taking one control plane implementation and using it to steer different data plane implementations. This is useful because it prevents vendor lock-in (at least from the hardware side). In addition, network nodes have become much simpler and inexpensive, as they do not have to deal with complicated and distributed information and decision-making (control plane). Moreover, OpenFlow can accelerate innovations in network services, as it is easier to prototype them in software.

OpenFlow is currently deployed in a wide range of networks such as campus networks, data centre networks, wide-area networks (e.g., Google B4), access networks (e.g., in access points) and so on. Recently, there is significant interest from research communities to apply OpenFlow in wireless MANET [5], [6]. One of the major drivers of OpenFlow is its simplification. It simplifies networks by allowing it to decouple complex software from nodes and deploying it in external servers called controllers.

One of the challenges behind deploying OpenFlow in wireless ad hoc networks is that available software for OpenFlow such as Open vSwitch[1] and ofsoftswitch13[2] are designed for wired networks. They partially support wireless ad hoc MAC (media access control) protocols [7]. In this paper, OpenFlow is deployed using current OpenFlow software (i.e., Open vSwitch) in wireless MANETs. In addition, this paper proposes a method with which OpenFlow can be configured automatically in MANETs (without any manual configurations). The challenge is that MANETs are very dynamic networks where network nodes may move with a different speed and direction. The speed and direction may also change at any time. Therefore, without deploying OpenFlow automatically, it may be difficult to implement MANETs with OpenFlow.

The idea is to deploy one or more controller nodes in the wireless ad hoc networks and to control the network from those controller nodes. In the proposed method, each wireless node in an ad hoc network discovers the controller automatically and establishes an OpenFlow session with it. We consider an ad hoc network in which the controller is directly reachable to only a few wireless nodes in the network. This research addresses the challenge for nodes that can not directly reach the controller and where they have to find a path to the controller through other nodes (mobile) in the network. This type of an ad hoc network is different from an in-band OpenFlow network, discussed for wired networks in [8]. This is because nodes in MANETs can move from one location to another.

The proposed configuration method is tested in an emulated MANET, created on the Fed4FIRE testbed. The MANET is created using Mininet-WiFi, which is an emulator for

[1]http://www.openvswitch.org
[2]http://cpqd.github.io/ofsoftswitch13/

software-defined wireless networks. Experimentation includes automatic configuration of OpenFlow in linear, sparse, and dense mobile ad hoc network. The results show that an OpenFlow session can be established automatically with the controller in minimal time. The results of mobility scenarios show the effectiveness of the innovative configuration method in re-establishing an OpenFlow session when wireless nodes or the controller move from one location to another. In addition, data traffic experiments were conducted, which measured the performance of OpenFlow enabled ad hoc networks.

The remainder of this paper is organised as follows: Section II describes problems with current OpenFlow software and the proposed innovative automatic configuration method. Section III provides the detail about emulations. The results are provided in Section IV. Section V presents the related work and finally, Section VI concludes the paper.

II. Automatic Configuration Proposal

The method for automatic configuration may differ with the types of OpenFlow nodes used in an ad hoc network. There are two types of OpenFlow nodes available in the market namely, Pure OpenFlow and Hybrid OpenFlow [9]. Pure OpenFlow nodes support only OpenFlow operations for forwarding packets. Hybrid OpenFlow nodes support both OpenFlow and traditional switching operations such as layer 2 switching and layer 3 routing for forwarding packets. This research proposes to implement the automatic configuration method in hybrid OpenFlow nodes where nodes implement both traditional routing and OpenFlow protocols.

For the proposed method, each hybrid OpenFlow wireless node contains two types of forwarding tables: (1) FlowTables of an OpenFlow switch and (2) traditional routing table. Data traffic is matched against the forwarding entries of the FlowTables of OpenFlow switches. Control traffic (e.g., traffic sent between the controller and OpenFlow nodes) is matched against the routing entries of the routing table.

A. Problems and Solutions

The MAC addressing scheme of current OpenFlow software such as Open vSwitch is based on the IEEE 802.3 standard. The problem for MANETs is that wireless ad hoc nodes do not support the IEEE 802.3 standard. However, they do support the ad hoc mode of the IEEE 802.11 standard. Therefore, the IEEE 802.3 frame (without transforming to the IEEE 802.11 frame) cannot be successfully transmitted or received over the wireless link. This problem is extensively studied by M. Rademacher et. al. in [7].

Currently, the following two options are explored to successfully transmit (or receive) the frame generated by the forwarding table of Open vSwitch on a wireless link: (1) performing MAC rewriting according to the IEEE 802.11 standard before transmitting a packet on a wireless link [17]. (2) supporting the 4addr mode [7] on the wireless interface. In the former case, an additional MAC rewriting component is required for packet header modification. In the latter case, the wireless interface driver will correctly replace the MAC addresses of the IEEE 802.3 (Ethernet) frame with the IEEE 802.11 format. The problem is that not all wireless nodes support the 4addr mode of the IEEE 802.11 standard.

This paper proposes to use a third option uses tunnels such as GRE or VXLAN to successfully transmit (or receive) a frame generated by the forwarding table of Open vSwitch on a wireless link. VXLAN stands for Virtual eXtensible Local Area Network and GRE stands for Generic Routing Encapsulation. Using these tunnels, a GRE or VXLAN header is inserted into an IEEE 802.3 packet coming from Open vSwitch. The MAC addressing scheme of these GRE packets depends on the underlying transmission medium (in this case it is IEEE 802.11).

Current OpenFlow software such as Open vSwitch support GRE and VXLAN tunnelling. Tunneling with Open vSwitch allows a wireless node to encapsulate traffic coming from Open vSwitch data-path (e.g., Ethernet traffic) with GRE or VXLAN header to transport over a wireless link. In order to create a GRE or VXLAN tunnel between a node and its neighboring node, the IP address of the neighbor node is required. In our automatic configuration method, the IP address of the neighboring node is known from the neighbors discovered by the routing protocol (see the next subsection). This approach introduces an extra overhead of 38 bytes in case of GRE in each hop. The 38 bytes comprises of 4 bytes for the GRE tag, 20 bytes for the IP header and 14 bytes for the Ethernet header per packet. More information on creating GRE or VXLAN tunnels using OpenvSwitch can be found at [11].

In the proposed method, data traffic follows the path through the tunnels, which are established to send (or receive) the traffic matched through the FlowTable of OpenFlow. However, control traffic is sent as usual, and not through the tunnels.

B. Assumptions, Requirements and Adapted Solutions

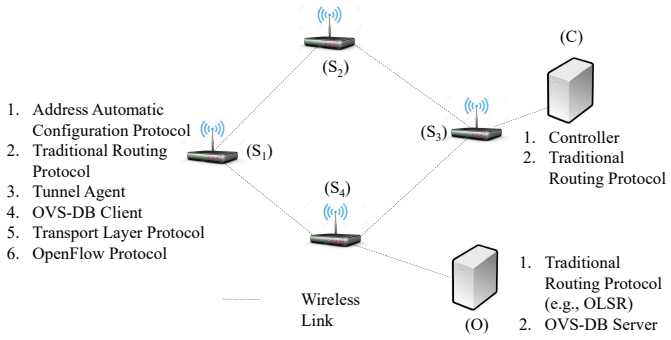

Fig. 1. A Wireless Ad hoc Network. S_1, S_2, S_3, S_4 are wireless network nodes, C is the controller and O is the OVS-DB server

It is assumed that wireless nodes in the considered wireless ad hoc network are connected (see Fig. 1). This means that each wireless node is reachable to all other nodes in the network either directly or via other nodes in the network (see S_1, S_2, S_3, S_4 in Fig. 1). Only a few nodes are directly reachable to the controller (see node S_3 in Fig. 1).

In the proposed automatic configuration method, each wireless node establishes an OpenFlow session with the controller on the top of a transport layer session (without any manual configuration). Each node establishes tunnels with its direct neighbour nodes to transmit or receive data traffic (i.e., matched against the FlowTables of OpenFlow wireless nodes). In order to implement the proposed automatic configuration method, we frame the following requirements:

1) The wireless node needs to configure a unique IP address to itself.
2) The wireless node needs to know the IP address of each of its neighbour nodes in order to create GRE or VXLAN tunnels between the node and neighbour nodes.
3) The wireless node needs to configure tunnels with its direct neighbours.
4) The wireless node needs to know the IP address and transport layer parameters such as the port of the controller.
5) The wireless node needs to know a path to the controller. The path may be through other nodes in the network, see Fig. 1.

The first requirement is solved by running an address auto-configuration protocol in each node [19]. Using the auto-configuration protocol, each node gets an IP address without running the DHCP (Dynamic Host Configuration Protocol) server in the network. For the second requirement, each node runs a traditional routing protocol in the network. An example of a traditional routing protocol is the OLSR-Optimized Link State Routing. For the third requirement, a tunnel agent is executed in each node. This tunnel agent creates a tunnel such as GRE or VXLAN with a neighbour node when the routing protocol discovers the neighbour. In addition to creating a new tunnel, the tunnel agent deletes an old tunnel with a neighbour which is no longer the neighbour of the node as detected by the routing protocol.

For the fourth requirement, OVS-DB (Open vSwitch Database management) server (RFC 7047) is deployed in the network (see Fig. 1) and an OVS-DB client is ran in each wireless node. Like the controller node, this server is also reachable to a few wireless nodes in the network (see S_4 in Fig 1). The OVS-DB server contains information about the controller node such as the controller IP address and other transport layer parameters. The OVS-DB server configures this information in each node by communicating with the OVS-DB client using the OVS-DB protocol. Running the routing protocol in the controller node fulfills the fifth requirement. Therefore, each node knows a path to the controller through the routing protocol.

C. Proposed Automatic Configuration Method

The proposed automatic configuration is described using one controller in the network (Fig. 1). However, the explanation will be similar for the network where there are more than one controllers in the network. Placement of the controllers is another problem in a wireless ad hoc network. However, this problem is not investigated in this paper.

Fig. 1 shows the network of four wireless nodes (S_1, S_2, S_3, S_4), an OVS-DB server and controller. Each wireless node (S_1, S_2, S_3, S_4) runs the following protocol stack: (1) an address auto configuration protocol [19], (2) traditional routing (e.g., OLSR), (3) the tunnel agent, (4) the OVS-DB client, (5) a transport layer protocol, and (6) the OpenFlow protocol. The OVS-DB server and controller also run the routing protocol.

1) Initial Setup: Each network wireless node (S_1, S_2, S_3, S_4) gets an IP address using an address auto-configuration protocol. As each node also runs a routing protocol, a node gets the information about a neighbour node (e.g., IP address), once the routing protocol discovers it. On reception of neighbour information, the tunnel agent running on the node creates a tunnel between the node and a discovered neighbour. The OVS-DB server configures the controller information (the controller IP address and transport layer parameters) in a wireless node, once the routing protocol running on the OVS-DB server discovers the wireless node in the network. A path between the OVS-DB server and the discovered wireless node is decided by the routing protocol.

Once a wireless node (S_1, S_2, S_3, S_4) knows the IP address of the controller (from the OVS-DB server), it runs ARP to know the MAC address of the controller. The ARP messages follow a path to the controller, decided by the routing protocol. After knowing the MAC address of the controller (using ARP), the wireless node establishes a transport layer session with the controller. The path for establishing the transport layer session is also given by the routing protocol. In the next step, the wireless node (S_1, S_2, S_3, S_4) establishes an OpenFlow session with the controller (using the same path used by the transport layer protocol).

2) Movement Scenario: Wireless nodes (S_1, S_2, S_3, S_4) including the controller and OVS-DB server may become unreachable with their neighbour nodes (discovered previously by the routing protocol) as they move from one location to another in a MANET. When the path between the controller (or the OVS-DB server) and a wireless node contains an unreachable neighbour, the communication between the controller (or the OVS-DB server) and the wireless node does not work until a new valid path (discovered by the routing protocol) is established in the network. In case the new valid path is established before the transport or OpenFlow layer (note that the OpenFlow layer is on the top of the transport layer) detects a communication failure, the previously established OpenFlow session with the node does not break and communication between the controller and the node starts to work again after the correct routing entries (of a valid path) are inserted in the network. Moreover, if the transport or OpenFlow layer detects a communication failure before the new valid path is inserted into the network, the previously established OpenFlow session between the node and the controller breaks and the new session is established when the correct routing entries (to reach the controller) are established in the network.

In addition to the above changes, when the routing protocol running on a wireless node (S_1, S_2, S_3, S_4) detects

an unreachable neighbour, the tunnel agent (running on the same node) deletes the tunnel (GRE or VXLAN) with the unreachable neighbour. When the routing protocol detects a new neighbour, the tunnel agent creates a tunnel with that neighbour. Moreover, the IP address of a wireless node can also change when moving from one location to another [19]. In this case, the previously established OpenFlow session with the controller is broken and a new OpenFlow session is established.

If more than one controller is used to control the network (not shown in Fig. 1), the OVS-DB server can update the node with the correct controller information when the controller information changes. This may be due to change in the location of a node or the controller. If the controller information changes, the node breaks the old OpenFlow session and establishes a new OpenFlow session with the new controller. A path to the new controller is decided by the routing protocol. Note that this is the responsibility of the OVS-DB server that which controller will be assigned to a node and is out of scope for this work.

III. Emulation Scenario

The emulations are performed on a node of the virtual wall testbed facility at IMEC, Gent, Belgium, provided by the Fed4Fire testbeds facility[3]. A pcgen04 node is chosen for the emulation of a wireless ad hoc network. This node has 8 CPU cores of Intel E5-2650v2 processor with 2.6 GHz speed, 48GB RAM and 1x 250GB hard disk. Hyper-threading is enabled in this node. A single physical CPU core with hyper-threading in this node appears as four logical CPUs to an operating system. Therefore, there are 32 logical CPUs in this node.

Mininet-WiFi[4] is deployed in the above pcgen04 node to run wireless ad hoc network emulations. Mininet-WiFi is a fork of the Mininet SDN network emulator[5] and extends it by adding virtualized wireless stations and access points based on the standard Linux wireless drivers and the IEEE 802.11 wireless simulation (i.e., mac80211_hwsim) driver. The ubuntu 16.04 LTS image (available through the Fed4Fire interface) is deployed on a pcgen04 node (n064-10) and then a Mininet-WiFi is ran on this node. An Open vSwitch[6] version 2.5.5 is installed in the node. Open vSwitch already has the OVS-DB protocol implemented in its software. The OVS-DB protocol is run on the top of the TCP (transmission control protocol). In addition, the POX controller[7] (version 0.5.0) is used for OpenFlow emulations in this experiments. In this emulation, one controller is used for emulation and the OVS-DB server is also located at the controller node. For assigning the IP addresses to each wireless node, we used the automatic IP assignment method available in Mininet-WiFi.

OpenFlow checks the aliveness of its session by sending a probe message (i.e., ECHO_REQUEST) and receiving a reply (i.e., ECHO_REPLY). If a reply is not received after sending three requests, a failure is declared and the session is broken. In this experiments, the ECHO_REQUEST interval is 5 seconds. Therefore, the OpenFlow session failure is detected in about 15 seconds. TCP is used as a transport layer protocol for communication between wireless OpenFlow ad hoc nodes and the controller. In addition, OLSR is used as a routing protocol in this network. OLSR version 0.6.8[8] is used for this emulation. The default values of emission parameters of OLSR given by its version 0.6.8 are used. The following are the default values: the hello interval is 2 seconds, the Traffic Control (TC) send interval is 5 seconds, and the host and network association (HNA) interval is 5 seconds. The neighbour hold time, TC hold time and HNA hold time are 10 times the hello interval, 60 times the TC send interval and 60 times the HNA interval, respectively.

A separate CPU is dedicated to each wireless node in the network including the controller and the OVS-DB server. The following types of ad hoc networks are emulated: (1) linear, (2) sparse and (3) dense network. In the linear network, a linear network is formed using 20 wireless nodes. The distance between a node and its neighbour is 40 meters and the radio range of each node is 74 meters. Each node contains one wireless interface (wlan0) which supports the ad hoc mode of IEEE 802.11. The transmission power is 14 dBm and the frequency is 2.412 GHz. The controller and the OVS-DB server is ran at the last node of the network and the configuration time is calculated.

In the sparse ad hoc network, 20 wireless nodes are deployed randomly over 250×250 meter square area. However, in the dense ad hoc network, 20 wireless nodes are deployed randomly over 125×125 meter square. All other parameters like radio range, transmission power and frequency are the same as the linear network. The controller and the OVS-DB server are run at one of the wireless nodes in the sparse and dense network. The automatic configuration time of these networks is then compared with the results gathered through the linear network.In addition, to show the effect of movement, the nodes in the network are moved to a different location and the controller re-connection time is calculated. Furthermore, the data traffic experiments are performed in the considered ad hoc network where the performance (in terms of the end-end delay) is calculated. This performance is then compared with the ad hoc network in which OpenFlow is not deployed.

IV. Emulation Results

OpenFlow session establishment time (in bootstrapping), OpenFlow reestablishment time (in mobility scenarios) and data traffic performance results are presented in this section.

Fig. 2 shows the time taken by a wireless OpenFlow node to establish an OpenFlow session with the controller after it boots up. This time is called the OpenFlow session establishment time. This time includes: (1) the time to run Open vSwitch software (2) the time to run the OVS-DB client, (3) the time

[3]https://www.fed4fire.eu/testbeds/virtual-wall/

[4]https://github.com/intrig-unicamp/mininet-wifi

[5]http://mininet.org/

[6]https://www.openvswitch.org/

[7]https://github.com/noxrepo/pox

[8]http://www.olsr.org/?q=download

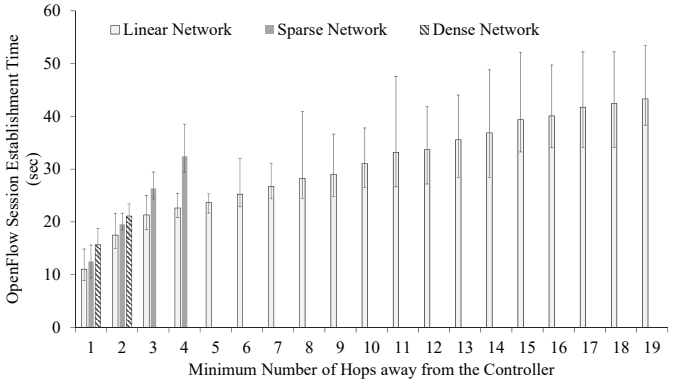

Fig. 2. OpenFlow Session Establishment Time. Error bars show the minimum, average and the maximum values.

to run OLSR (including the OLSR configuration time), (4) the time to run the OVS-DB server, (5) the time to configure required parameters for an OpenFlow session (e.g., controller IP address) and (6) the time to establish an OpenFlow session after getting the controller parameters.

This time is calculated for the nodes which are at a certain hop (hop 1 to hop 19 in Fig. 2) away from the controller. This time is calculated for different types of topologies: linear, sparse and dense. For the topologies where there is no node at a certain hop from the controller, no bar is shown in Fig 2. For example, there is no bar in the sparse network in Fig. 2 after hop 4 and there is no bar in the dense network after hop 2. As in the sparse and dense network, there may be many nodes at a certain hop from the controller, the average value of the OpenFlow session establishment time is calculated. The results are calculated 50 times and the minimum, average and maximum values are shown in Fig. 2.

Fig. 2 shows that 1 hop nodes from the controller take a significantly long time (approximately 10 to 15 seconds) to establish an OpenFlow session with the controller. After 1 hop, the OpenFlow establishment time increases either linearly or non-linearly as shown in Fig. 2. In Fig. 2, for some nodes (e.g, hop 1 to hop 4 and hop 17 to hop 19 in the linear network) there does not appear to be a linear increase in the OpenFlow session establishment time with an increase in the number of hops from the controller. This is because even when the OVS-DB server discovers a node in the network (through the routing protocol), there may be a case that the routing entries along the path to the node are not yet established in the network to forward traffic. The OVS-DB server sends a TCP Syn message to establish the TCP session when it discovers a new node in the network. The TCP Syn message reaches to the node if all the nodes along the path to the node have the routing entries to reach the node. In the case where the TCP Syn message does not reach the node, the OVS-DB server backoffs for 1 second and again sends a TCP Syn after backoff. However, if the TCP Syn again does not reach the node, it follows exponential backoffs. In the exponential backoff, the backoff doubles on each unsuccessful transmission. This is the reason that for some hops we do not see a linear relationship between the number hops and the OpenFlow session establishment time.

The OpenFlow establishment time in the dense network is longer than in the sparse network. This is because in the dense network, a greater number of nodes are available at a certain hop from the controller and therefore, a large number of OpenFlow session paths need to be established for the certain hop. However, the OpenFlow establishment time is shorter in the linear network than in the sparse network. The reason is the same as explained before for the dense and sparse network.

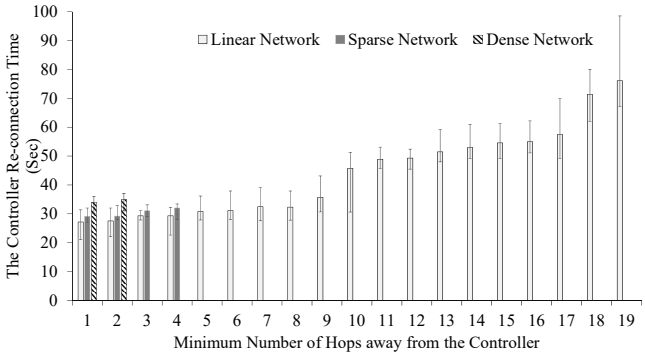

Fig. 3. Controller Re-connection Time. Error bars show the minimum, average and the maximum values

Fig. 3 shows the time to reconnect the controller when one or more nodes in a path to the controller move to another location. Like Fig. 2, the results are calculated 50 times and the minimum, average and maximum values are shown in Fig. 3. Like Fig. 2, no bar is shown in Fig 3 for the topologies where there is no node at a certain hop from the controller. OpenFlow in this experiments detects a failure (due to a node in a path to the controller moves to another location) after 15 seconds and then it breaks the session with the controller. This automatically breaks the TCP session. The node then also tries to reestablish the TCP session by sending a TCP syn and performs backoff (as described in the previous paragraph). In this case, OLSR detects the failure after 20 seconds and tries to find a new path to the controller. Therefore, the controller re-connection time in Fig. 3 is more than 20 seconds.

In this mobility scenarios, all the nodes move from one location to another. Therefore, all the nodes along a path to the controller have to establish new routing entries to reach the controller or other nodes in the network. If a node reaches a location which is a few hops away from the controller, the path is established in a short time. However, if the location is very far from the controller, the path is established in a long time, resulting in an increase in the controller re-connection time.

Fig. 4 shows the result of data traffic experiments in which data traffic is sent from a node to each and every other node in the network. UDP packets are sent at the interval of 100 milliseconds over 2 minutes and the minimum, average and maximum value of the end-to-end delay is calculated. The packet size was 1000 bytes and the linear network was considered for emulations. No separate queues are implemented for data and control traffic. They share the same channel. After

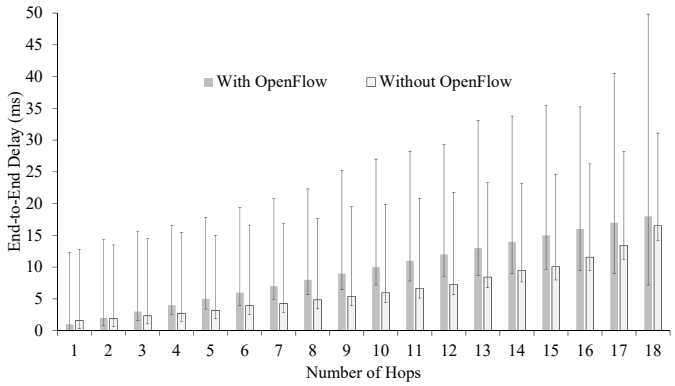

Fig. 4. Data Traffic Performance. Error bars show the minimum, average and the maximum values

the hello interval each node has to send hello messages to its neighbour nodes and then network information has to be broadcasted over the network. It was observed that this has an impact on data traffic performance. In addition, the first packet always takes a long time to reach the destination node, as nodes have to run ARP to know the MAC address of the destination node before sending the first packet. This is the reason for the variation in the maximum and minimum value of the end-to-end delay in these experiments.

Fig. 4 shows that the average value of end-to-end delay increases as the number of hops between the source and destination increases. This is because data traffic has to travel more number of hops to reach the destination. In addition, it shows that the end-to-end delay is longer for a network deploying OpenFlow than the network in which OpenFlow is not deployed. This is because tunnelling is used to transport data traffic generated using Open vSwitch software (following the IEEE 802.3 standard) on a wireless link. The performance degradation using OpenFlow happened because we used tunnelling in nodes to transport OpenFlow traffic on wireless links. The average value of the performance degradation (in form of the average delay) is low as 1 ms (see Fig. 4) when the number of hops travelled is small (i.e., < 6).

V. Related Work

OpenFlow is a communication protocol that provides access to the forwarding plane of a network switch or router over the network [9]. One of the first deployments of OpenFlow in wireless networks was at the Stanford campus network [10] where OpenFlow was deployed at WIFI-Acess points (AP) and WiMax base stations. In [12], a practical implementation of an SDN (Software Defined Networking) MANET testbed was provided. This work designs a testbed that has the advantages of device-to-device data communication and simplicity of network configuration using a centralized controller. In [13], an attempt was made to implement OpenFlow over a wireless mobile ad hoc network of smartphones. It is shown that the development time and complexity of an application can be significantly reduced by using OpenFlow in a wireless ad hoc network.

Several advantages of SDN/OpenFlow in tactical wireless ad hoc networks are discussed in [14]. It is shown that using SDN/OpenFlow, it is relatively simple to configure and reconfigure wireless ad hoc nodes based on dynamic policies, mission-critical objectives and battlefield condition changes. A detailed performance analysis of SDN and OpenFlow are presented in [15] for wireless networks through simulation results using an OMNeT++ network simulator. The authors in [16] addresses the issues of low transfer rate and small coverage of meshed ad hoc networks by using a combination of BATMAN (Better Approach To Mobile Ad-hoc Network) routing protocol, Open vSwitch, and Dijkstra algorithm. In [18], the Parallel Redundancy Protocol (PRP) is used using OpenFlow/SDN paradigm in Wireless Local Area Networks to reduce the fail-over time. The main critique of the existing work is that they do not discuss the automatic deployment of OpenFlow without any manual configuration in the context of MANET or any other wireless networks.

VI. Conclusions and Future Work

This paper proposes an innovative method that can automatically configure OpenFlow in wireless mobile ad hoc networks. The method can be implemented in hybrid OpenFlow nodes which support both the traditional routing and OpenFlow protocol. This research is the first work on the automatic configuration of OpenFlow in wireless ad hoc networks.

Extensive experiments were performed on different types of wireless ad hoc networks (linear, sparse and dense) generated on the Fed4Fire testbed using Mininet-WiFi. The results show that the proposed innovative method is able to automatically configure a wireless ad hoc network of 20 nodes in less than 40 seconds. In addition, mobility scenarios were considered in which each node (including the controller) moves from one location to another. The controller re-connection time was calculated in these scenarios. Moreover, data traffic experiments were performed. The data traffic experiments show that the performance of the network deploying OpenFlow is lower than the network in which OpenFlow is not deployed. This is because tunnelling was used to transport data traffic generated using OpenFlow software (following the IEEE 802.3 standard) on a wireless link. Future work could focus on decreasing this performance degradation by possibly testing with the other two options discussed in Section II-A, in this current framework.

Based on the presented emulation results, future work can involve applying the innovative automatic configuration method to production networks. However, to improve the accuracy of results, experiments can also be performed on real environment testbeds [20] such as w-iLab.t facility and IRISH testbed facility at the Fed4Fire testbed where each ad hoc node (including the controller and OVS-DB server) is a wireless node (instead of a node generated by Mininet-WiFi software). A part of this work is lively demonstrated using a portable testbed (two laptops) at [21].

Security issues of OpenFlow in a wireless ad hoc network have not been explored in this research. Many of security issues and their solutions using SDN-MANET are explored

in [22]. Future work may explore the security space of OpenFlow-based MANETs using the concepts of Blockchain.

ACKNOWLEDGEMENT

Authors would like to thank the team of Fed4Fire testbed, Gent, Belgium for providing the experiment facility and support to carry out the experiments performed in this paper.

REFERENCES

[1] T. Wang, J. Liu, L. Cheng, H. Xiao, "Robust collaborative mesh networking with large-scale distributed wireless heterogeneous terminals in industrial cyber-physical systems," International Journal of Distributed Sensor Networks, Vol. 13(9), 1-21, 2017

[2] T. Alam, M. Benaida, "The Role of Cloud-MANET Framework in the Internet of Things (IoT), " International Journal of Online Engineering, Vol. 14(12), pp. 97-111 -2018

[3] Zhou, B., Dastjerdi, A.V., Calheiros, R.N., Srirama, S.N. and Buyya, R., "A context sensitive offloading scheme for mobile cloud computing service," 2015 IEEE 8th International Conference in Cloud Computing", pp. 869-876, 2015.

[4] N. McKeown, T. Anderson, H. Balakrishnan, "OpenFlow: Enabling Innovation in Campus Networks, SIGCOMM Comput. Commun. Rev, 2008.

[5] K. Poularakis, G. Iosifidis and L. Tassiulas, "SDN-Enabled Tactical Ad Hoc Networks: Extending Programmable Control to the Edge," in IEEE Communications Magazine, vol. 56, no. 7, pp. 132-138, July 2018.

[6] S. Hadzic, C. Niephaus, O. G. Aliu, G. Ghinea, and M. Kretschmer, "Wireless Back-haul: a software defined network enabled wireless Back-haul network architecture for future 5G networks, IET Networks, vol. 4, no. 6, Nov. 2015, pp. 287295.

[7] M. Rademacher, F. Siebertz, M. Schlebusch and K. Jonas, "Experiments with OpenFlow and IEEE802.11 Point-to-Point Links in a WMN", Twelfth International Conference on Wireless and Mobile Communications, 2016.

[8] S. Sharma, D. Staessens, D. Colle, M. Pickavet and P. Demeester, "In-band control, queuing, and failure recovery functionalities for openflow," IEEE Network, vol. 30, no. 1, pp. 106-112, 2016.

[9] OpenFlow Specification, "SDN Technical Specifications". [Online]. Available:https://www.opennetworking.org/software-defined-standards/specifications/

[10] K. Yap, R. Sherwood, M. Kobayashi, T.-Y. Huang, M. Chan, N. Handigol, N. McKeown, and G. Parulkar, "Blueprint for Introducing Innovation into Wireless Mobile Networks, in ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures, 2010, pp. 2532.

[11] Open vSwitch Tunnels: http://docs.openvswitch.org/en/latest/howto/tunneling/

[12] H. C. Yu, G. Quer and R. R. Rao, "Wireless SDN mobile ad hoc network: From theory to practice," 2017 IEEE International Conference on Communications (ICC), Paris, 2017, pp. 1-7.

[13] P. Baskett, Y. Shang, W. Zeng and B. Guttersohn, "SDNAN: Software-defined networking in ad hoc networks of smartphones," 2013 IEEE 10th Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, 2013, pp. 861-862.

[14] K. Poularakis, G. Iosifidis and L. Tassiulas, "SDN-Enabled Tactical Ad Hoc Networks: Extending Programmable Control to the Edge," in IEEE Communications Magazine, vol. 56, no. 7, pp. 132-138, July 2018.

[15] G. Araniti, J. Cosmas, A. Iera, A. Molinaro, R. Morabito and A. Orsino, "OpenFlow over wireless networks: Performance analysis," 2014 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting, Beijing, 2014, pp. 1-5.

[16] S. Koh, J. Kim and S. Lee, "A proposal of OpenFlow controller to improve transfer rate in mesh network," 2017 International Conference on Information Networking (ICOIN), Da Nang, 2017, pp. 509-511.

[17] A. Detti, C. Pisa, S. Salsano, and N. Blefari-Melazzi, "Wireless Mesh Software Defined Networks (wmSDN), in Wireless and Mobile Computing, Networking and Communications (WiMob), 2013 IEEE 9th International Conference on, Oct. 2013, pp. 8995.

[18] E. Molina, E. Jacob, A. Astarloa, "Using OpenFlow to control redundant paths in wireless networks", Network Protocols and Algorithms, Vol. 8(1), 2016

[19] A. Munjal, "Address Auto-Configuration Protocols and their message complexity in Mobile Adhoc Networks," PhD Dissertation, IIT Kanpur, 2015

[20] M.Berman et. al., "Future Internets Escape the Simulator", Communications of the ACM, Vol. 58(6), pp. 78-89, 2015

[21] S. Sharma, M. Nekovee, "Demo Abstract: A demonstration of automatic configuration of OpenFlow in wireless ad hoc networks", IEEE INFOCOM, Demonstration Session, April 2019

[22] M. Alqallaf, "Software Defined Secure Ad Hoc Wireless Networks," PhD Dissertation, Wright State University, 2016