A Simulation Environment for Software Defined Wireless Networks with Legacy Devices

Daniela Sousa
Instituto de Telecomunicações, 3810-193 Aveiro
University of Aveiro, 3810-193 Aveiro
Portugal
dcsousa@ua.pt

Susana Sargento
Instituto de Telecomunicações, 3810-193 Aveiro
University of Aveiro, 3810-193 Aveiro
Portugal
susana@ua.pt

Miguel Luís
Instituto de Telecomunicações, 3810-193 Aveiro
ISEL - Instituto Superior de Engenharia de Lisboa, 1959-007 Lisboa
Portugal
nmal@av.it.pt

ABSTRACT

The adoption of Software Defined Networks (SDNs) in a Mobile ad-hoc network (MANET) could present several benefits, such as adaptability and performance increase. However, to assess this possibility, a simulation tool may be necessary to test new protocols and solutions in a large combination of scenarios and traffic patterns, without the need of real equipment. Unfortunately, few tools are available for wireless SDNs, and none have the ability to also support MANETs with multiple radio access technologies. While NS-3 has the ability to simulate heterogeneous MANETs, it does not support wireless OpenFlow capable devices or wireless OpenFlow channels. In this work we present a simulation environment that, besides creating an *ad-hoc* data plane, enables the possibility of creating wireless hybrid SDN devices capable of connecting to legacy devices, alongside with an LTE OpenFlow channel connected to an external SDN Controller (RYU). Results show that the simulation environment supports large networks with both legacy and SDN devices, although these will bear an effective running time higher than their simulation time. Moreover, when comparing to an OLSR-only network, the proposed network (with a basic path search metric) has the same or higher performance.

CCS CONCEPTS

• **Networks** → **Network simulations**; **Hybrid networks**; **Programmable networks**; **Mobile ad hoc networks**.

KEYWORDS

Software Defined Network, Mobile Ad-hoc Network, Network Simulator 3, Wireless Hybrid SDN devices

ACM Reference Format:
Daniela Sousa, Susana Sargento, and Miguel Luís. 2022. A Simulation Environment for Software Defined Wireless Networks with Legacy Devices. In *Proceedings of the 18th ACM International Symposium on QoS and Security for Wireless and Mobile Networks (Q2SWinet '22), October 24–28, 2022, Montreal, QC, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3551661.3561369

Q2SWinet '22, October 24–28, 2022, Montreal, QC, Canada

ACM ISBN 978-1-4503-9481-9/22/10…$15.00
https://doi.org/10.1145/3551661.3561369

1 INTRODUCTION

SDNs have been studied for many years promising a more efficient, dynamic and programmable network management compared to the *traditional* routing protocols [11]. Still, the majority of the works published in the literature are focused on wired environments, *i.e.* the SDN domain is restricted to non-mobile network nodes [15].

The adoption of SDN in Wireless Local Area Networks (WLANs) is inevitable, as they are one of the most common access technologies. Moreover, SDN in WLANs can have several benefits, such as efficient network management, unified management of wired and wireless networks, unplanned and uncoordinated deployment, flexible control, and improvement of quality of service [28]. In WLANs, several access technologies can share the spectrum. By removing the control from the devices, the controller can gain a global view, allowing a dynamic selection of data transmission schemes, efficient allocation of radio resources, and better handover policies, thus, improving the use of bandwidth resources and Quality of Service (QoS) [24].

Among the several WLANs, we have Mobile Ad-Hoc Networks (MANETs): their elements are capable of moving around, creating connectivity links with other mobile elements and/or with the infrastructure - through Access Points, also denoted as Points of Attachment - for extended connectivity, including the Internet. Amongst the several scenarios where these networks can be useful, one can enumerate vehicular networks for the dissemination of traffic-related information and infotainment purposes [36], or even in more critical scenarios such as disaster recovery or military operations [15, 33].

In MANETs, it is common for a single node to not have all the information about the network in order to determine if a given path has sufficient resources for a certain QoS or to perform load balancing features, to name a few. This limitation is sometimes related with the decentralized nature of these networks or with the energy-constrained characteristic of its elements. Therefore, it is suitable to have a global point of view capable of taking control decisions. The SDN approach can be used to overcome this problem by gaining global knowledge of the available topology, state, and requirements, providing a better QoS [2]. Still, the introduction of the SDN paradigm in such networks is far from being an easy

task. From one side, these networks are frequently built with heterogeneous devices, and therefore, the network needs to support legacy and SDN-based devices. On the other hand, the introduction of a centralized control (SDN controller) still remains a challenge in MANETs, where many works believe that the benefits brought by the SDN centralized control are worth the application to such networks [15].

Simulation or emulation platforms are a tool to facilitate the evaluation of network-related mechanisms and new approaches. Several platforms exist for SDNs and OpenFlow, being the most common ones NS-3 [30] and MiniNet [16]. However, only NS-3 supports the deployment of mobile wireless nodes with multiple radio access technologies. Moreover, as previously referred, in a real *ad-hoc* scenario, not all devices may be SDN capable, as devices are chosen opportunistically as they are available. Therefore, a simulation approach should account for the existence of legacy devices in the MANETs. Furthermore, the most widely accepted southbound interface, OpenFlow, does not support wireless environments [28], which is incompatible with an *ad-hoc* environment.

In this paper we propose an NS-3 SDN and wireless simulation environment capable of supporting different technologies, in a wireless *ad-hoc* data plane with both legacy and SDN devices. Moreover, this platform supports different technologies, such as WiFi and cellular networks. This simulation platform is tested in a WiFi mobile ad-hoc network, with legacy and SDN-capable devices, a cellular OpenFlow channel, and an external Controller. Results show that the platform can be scalable, and when compared to an OLSR-only network, the proposed solution can have the same or better performance while not taking full advantage of the centralized control. Moreover, this work will be made available to the NS-3 community for the support, for the first time, of legacy and SDN nodes in the same scenario.

This paper is organized as follows. The related work is discussed in Section 2. The proposed approach for the simulation of spontaneous wireless hybrid SDN, in both wireless ans cellular networks, is addressed in Section 3 (architecture) and 4 (controller and Openflow channel). The simulation performance is presented and discussed in Section 5. Finally, Section 6 concludes the paper and discusses future research directions.

2 RELATED WORK

Network simulators can be used to evaluate network approaches related to routing protocols, controller placement, scalability and others [6, 18]. The most popular open source network simulators are NS-2 [12], NS-3 [30], OMNET++; usually, NS-3 is the fastest simulator and is suitable for large networks [19].

Moreover, specifically for SDN environments, EstiNet [35] and MiniNet [16] are evaluated in [34], in which EstiNet has a better accuracy for larger networks, but lower time efficiency.

2.1 MiniNet

MiniNet supports fast prototyping using limited resources [21]. It is lightweight on virtualization features and network namespaces. It also allows to debug in real time, while NS-3 does not. The forwarding devices are OpenFlow switches, also known as open vSwitches, and the links are developed using Ethernet or Wi-Fi in the version Mininet-WiFi. However, it lacks any other technology support, and mobility modeling [4].

2.2 Hybrid (OpenNet)

Chan et al. [4] proposed a hybrid simulator, joining NS-3 and MiniNet. They integrate MiniNet controllers in NS-3 to overcome the lack of controllers compatibility and the incomplete handover process. This solution, called OpenNet, supports wireless networks given by NS-3 and controller compatibility, enabling handover in mobile nodes in different channels. However, it is not a maintained simulator, being the last release on September of 2017. Moreover, it is only compatible with Ubuntu version 14.04.5, that ends its life on 2022.

2.3 Network Simulator 3 (NS-3)

NS-3 has been created to support traditional network protocols; it also supports SDN. However, this official module only supports version 0.8.9 of the OpenFlow protocol [26]. Moreover, because real OpenFlow controllers are user-level programs, they cannot be compiled and linked with NS-3. Thus, only controllers specifically developed for NS-3 can be run [27].

Chaves et al. [7] developed an enhanced module called *OFSwitch13*, that includes all features of OpenFlow version 1.3[1], where External libraries are linked and compiled with NS-3 simulation engine in order to simulate the operations between forwarding devices and controllers. However, it does not support wireless technologies or hybrid routing mechanisms. Moreover, this module in NS-3 has several disadvantages: (1) at run-time, the manager cannot add or remove nodes from the simulation; (2) it has limited support for SDN controllers; and (3) in SDN, the handover process is not implemented completely. However, it also has the advantages of supporting several communication technologies, mobility patterns, protocols, and others, with the support of a big community, having several testbed platforms for wireless heterogeneous SDNs [8, 10].

Park et al. [14] proposed a testbed in NS-3 simulator for wireless heterogeneous software defined networks, where OpenDayLight is used as the controller with tap bridging to solve the support and realism regarding controllers. However, the source code is not public, and OpenDayLight only supports version 1.0 of OpenFlow [17].

Manjeshwar et al. [23] developed a SDN-based network architecture in NS-3 for unified control of multiple radio access technologies. Despite proving that this architecture improves the performance of the proposed approach, only the base stations are OpenFLow enabled devices.

Shastry et al. [32] uses a mesh network to connect legacy devices to wireless switches. However, the switch acts as a cluster head, and is connected by a wired link to the controller. Labraoui et al. [20] developed a NS-3 platform with out-of-band controller in an ad-hoc environment. The hybrid SDN-OLSR protocol proves to outperform three of the most widely used MANET routing protocols (AODV, DSDV, and OLSR). However, the architecture uses a mesh network, which many off-the-shelf devices may not support.

[1] http://www.lrc.ic.unicamp.br/ofswitch13/

2.4 Hybrid SDN

By having a MANET, which uses the already in-place devices, we can combine it with the advantages of an SDN architecture, therefore having a hybrid SDN without the need to replace the existing legacy devices [9]. A hybrid architecture can be achieved by (1) having hybrid SDN-capable devices [3], or (2) by deploying SDN switches as gateways in a legacy network [22]. However, this last approach increases the traffic and load in those switches, and creates points of failure. Moreover, in order for the SDN device to communicate with the legacy devices, it has to run a common routing protocol, making the first solution the most adequate for our problem. However, such device does not exist in NS-3.

Several works have tried to support SDN and cellular networks, or SDN and wireless environments in the several network simulators. However, to the best of our knowledge, works with multiple radio access technologies and the ability to support multi-hop forwarding through both legacy and SDN-based devices on a mobile ad-hoc scenario are still lacking. With this work, our contribution is an NS-3 simulation environment with WiFi and cellular technologies, capable of supporting both legacy and SDN network elements in an *ad-hoc* opportunistic environment.

3 SIMULATION ARCHITECTURE

Our approach supports multi-hop forwarding through both legacy and SDN-based devices in a mobile ad-hoc network with multiple radio access technologies. However, as stated before, NS-3 does not support SDN devices in a MANET context, hybrid SDN networks, or even a wireless OpenFlow channel.

NS-3 contains an SDN controller in its platform, the *ofswitch13*. Despite *ofswitch13* module is capable of supporting a newer version of OpenFlow, it does not support wireless technologies or hybrid routing mechanisms. A better approximation to the reality can be achieved by using an existent external controller, instead of a developed internal one. The *ofswictch13* module enables the connection to an external controller through a *tap* bridge interface. However, this functionality was developed for Floodlight 1.2 controller, and other newer controllers have not been tested or implemented. The *ofswitch13* module was built under the assumption that each SDN switch is connected to the controller by a Point-to-Point or Ethernet connection, which in a MANET cannot be supported, as the SDN-capable devices can also be mobile. Therefore, an wireless OpenFlow Channel needs to be implemented.

Figure 1 shows the architecture of the proposed simulation environment, where an external controller is connected to the NS-3 platform. Furthermore, the proposed simulation environment can run in a docker container, so that multiple instances can be ran simultaneously and independently from the computer's configurations. However, the proposed platform only supports one external controller, which is described in Section 4.1.

The proposed NS-3 simulation environment is composed of an LTE eNodeB, described in Section 4.2, several nodes (hence forth referred to as legacy devices), and several Hybrid SDN (H-SDN) devices, described in Section 3.1. To create a MANET, the data plane needs to work under *ad-hoc* mode, where each H-SDN device needs at least one wireless interface in this mode. Moreover, despite the original *ofswitch13* module supporting virtual interfaces, the

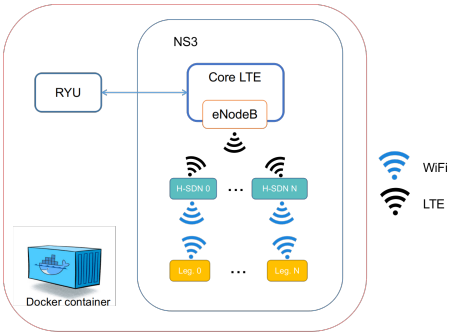

Figure 1: Proposed simulation environment - architecture overview.

proposed platform only works under the assumption that an OpenFlow port is mapped to a physical interface. In order for the SDN device to be connected to the legacy devices, a common routing and discovering protocol needs to be used.

Furthermore, in these networks, heterogeneous devices may exist with several different radio access technologies; however, each technology has an unique network address so that the controller can distinguish each interface's technology.

3.1 Hybrid Device

Figure 2 depicts how each module is connected to create the hybrid device in this SDN-enabled simulation environment. The blue modules and interfaces are the ones already existing in NS-3. The rose modules are the ones that had to be changed for the proposed simulation environment.

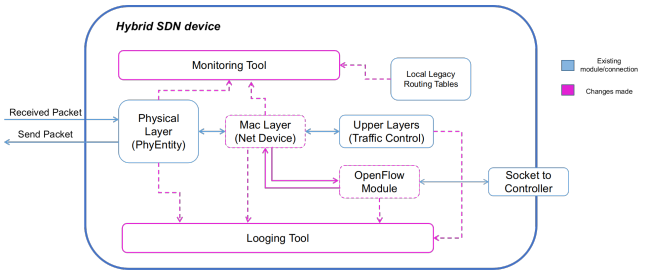

Figure 2: Hybrid Device architecture.

When a packet is received (except on the control interface), the monitoring tool gathers link measurements on both the physical and Medium Access Control (MAC) layers. Afterwards, the received packet, depending on the connection to the controller, can go to the *traffic control* module, which then forwards it to the upper OSI layers, or the *OpenFlow* module.

When the OpenFlow pipeline is selected, before entering the pipeline, a header change is performed if the packet was received in a Wi-Fi interface. This mechanism is further explained in Section 3.1.2.

Moreover, several features were added to the proposed node:

- 'Keep Alive' mechanism;
- Wireless OpenFlow ports;

- Workflow manager;
- Locally created packets are sent to the OpenFlow tables;
- LTE OpenFlow Channel;
- Monitoring Tool;
- Logging Tool.

3.1.1 ***OpenFlow Module: ofswitch13.*** As mentioned in Section 2.3, the *ofswitch13* library was used to implement the OpenFlow node. However, this module does not have support for wireless interfaces, wireless OpenFlow channels or any other hybrid mechanism[2]. Figure 3 shows the new developed class that works with the *ofswitch13* library. This new module has four main functions:

- `InstallSwitch`: Responsible for installing wireless interfaces as OpenFlow ports, along with the legacy routing protocol. Moreover, it also installs the Energy model (*BasicEnergySource*) on the node;
- `InstallExternalController`: *Hybrid OPF Device Helper* only supports calling this function if the Channel type is *SINGLECSMA* or *SINGLELTE*. textitSINGLECSMA is the default behaviour from the library. In textitSINGLELTE the LTE infrastucture, eNodeB, and the Controller are created.
- `CreateOpenFlowChannels`: If the Controller was created by the above function, it only allows the two mentioned Channel Types. In SINGLELTE, a new LTE interface is added to the nodes installed as switches. Afterwards, each node is connected to the eNodeB installed before.
- `SendStatsMsgtoCtrl`: This function is responsible for scheduling the periodic Experimental message that is sent to the controller, as will be described later in Section 5.1.

Changing the OpenFlow channel from a CSMA or point-to-point interface to a wireless interface creates the problem of connection loss, where an alternative behaviour needs to be implemented for the situation when the connection is broken. Each time an OpenFlow message is sent or received, the module sends a message to a callback stating that a message from or to the controller was sent. This starts the 'Keep Alive' mechanism, where the state machine instantiated in the *NetDevice* class will parse these messages to evaluate the connection to the controller.

Several changes were made to the *WifiNetDevice* class, as shown in Figure 4. These changes have to be performed on every wireless *NetDevice* that is mapped as an OpenFlow port.

3.1.2 ***Net Device Class.*** This class has a state machine with two states *BrokenConn* or *Connected*, where if the connection to the controller does not exist, the node behaves like a legacy device (state *BrokenConn*), and if there is one, it behaves like an SDN node (*Connected* state). This state machine is one of the components of the previously mentioned 'Keep Alive' mechanism. This mechanism requires the controller to send an *OpenFlow Echo reply* message after receiving five experimenter messages from the same H-SDN device. These replies act as an Acknowledge message to the experimenter messages. As stated before, the OpenFlow module sends a message to the `WorkflowMonitor` function, stating that a packet

[2]The *ofswitch13* module, on release 3.29, had an issue on the Ethernet 802.3 packet header parsing, whose solutions were published on the git repository of the ofswicth13 library (https://github.com/ljerezchaves/ofswitch13/issues/43). Moreover, other minor issues were fixed.

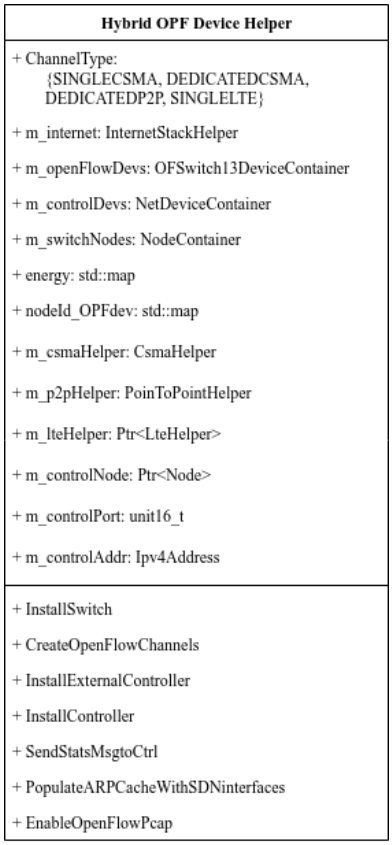

Figure 3: Hybrid device NS-3 helper.

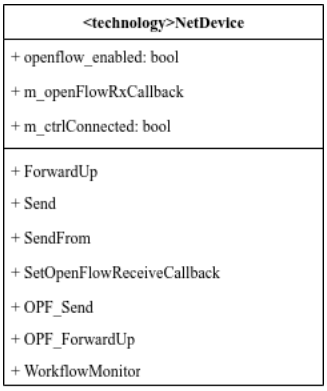

Figure 4: Changes to NetDevice module on NS-3.

was received or sent to the controller. When more than five experimenter messages are sent without any message being received, the connection is considered broken, passing to *BrokenConn* state. Afterwards, all packets are forwarded by the function `ForwardUp` to the upper layers until the connection is re-established. If the connection is up, the packets are sent to the OpenFlow module by the callback `m_openFlowRxCallback`.

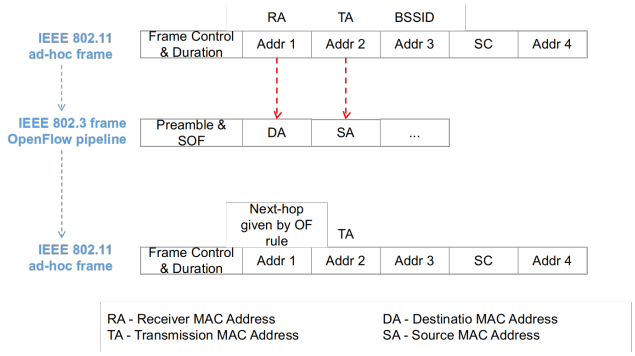

Figure 5: Header change scheme.

If the received packet's interface is a wireless one, a header change needs to be performed, so off-the-shelf OpenFlow solutions, like the *ofswitch13* library, can be used. This solution was adopted because it does not increase the overhead in the wireless link on the Data Plane (DP), like tunneling solutions [31]. Another advantage is that it does not require the *4addr* mode [29], which many off-the-shelf devices do not support, not restraining its adoption to real environments.

Figure 5 shows how the header change process is done for an exemplary IEEE 802.11 frame. The receiver and transmitter addresses are mapped to IEEE 802.3 source and destination addresses. After this packet is set to be transmitted through a wireless interface, the transmitter address is filled with the wireless interface's IP, and the receiver is given by the Rule on which this packet matched in the tables.

Another feature of the H-SDN device is the ability to forward created messages through the OpenFlow tables. Each message that is created and reaches the function `Send` or `SendFrom` is sent to the OpenFlow Callback, which then uses the function `OPF_Send` to send the message to the desired output interface.

Moreover, the *OPF_NORMAL* output port action was also implemented by the function `OPF_ForwardUp`, which receives the messages sent to this OpenFlow port and forwards them to the upper layers of the node to be treated like in a legacy device.

3.1.3 ***Monitoring Tool.*** This tool gathers device and link measurements in a passive way by the received messages, and in an active way by receiving packets sent by the controller from a neighbour, therefore making it a hybrid tool.

Functions `MonitorSniffRx`, `MonitorMacRx`, `MonitorMacTx`, shown in Figure 6, are the functions mapped to the trace sources from the Phy/Rx, MacRx and MacTx, respectively. They are then used to gather and calculate measurements like SNR, delivery ratio, bandwidth, and delay. These are in turn used to compile the experimenter message used by the topology discovery process in the controller (described in Section 5) along with information from the local routing tables.

3.1.4 ***Logging.*** The **logging module**, whose class is represented in Figure 7, is a tool that gathers measurements from the simulation's state every second, saving it to a file for later analysis.

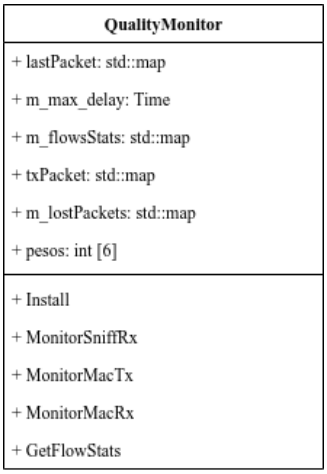

Figure 6: Quality monitor class.

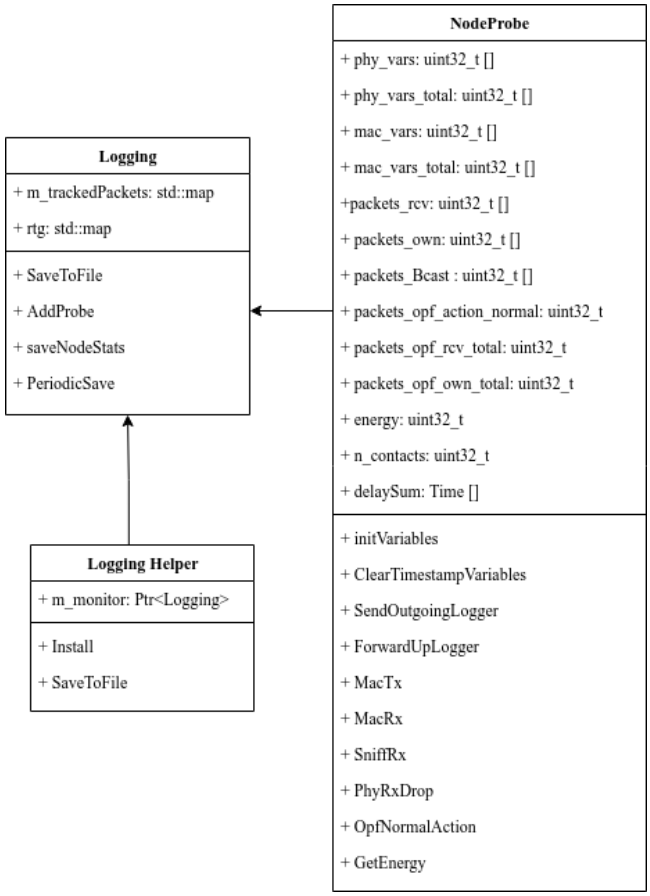

Figure 7: Logging Classes.

For each wireless interface mapped as an OpenFlow port, the module subscribes to the trace sources of the Received and Drop packets in the physical layer, and the received and sent packets in

the MAC layer. Moreover, it also subscribes to the trace sources of the packets output by the OpenFlow module to the legacy upper layers, and the received and created packets from the *Ipv4L3Protocol*. With all these functions, the module can gather measurements from the control channel (like, for example, how many OpenFlow packets the node has received), and from the wireless interfaces (like the number of packets received or dropped). However, in NS-3, we cannot differentiate from dropped packets by collision from the dropped packets from propagation loss.

For the delay variable, the MacTx Trace source on the legacy devices needs to be subscribed, so that the delay can be calculated. In a real network, the network protocol can timestamp the packets, so no modules need to be inserted in the legacy devices.

4 CONTROLLER AND OPENFLOW CHANNEL

4.1 Ryu Controller

As mentioned before, the *ofswitch13* module enables the use of external controllers to NS-3 through a *tap* bridge interface. A better approximation to reality can be achieved by using an existing (in use) controller.

Zhu et al. [37] compared most of the existing SDN controllers. From the more common controllers, both OpenDayLight and ONOS used Java as their programming language. However, there are a lot more applications written in Python available online. RYU and POX use such language, and RYU is one of the only controller that supports the newer versions of OpenFlow.

Therefore, despite the fact that RYU has a centralized architecture, it was chosen because of its popularity in the open source community, its versatility (applications can be rapidly developed by using python), and its support for OpenFlow version 1.3, which is compatible with the NS-3 *ofswitch13* module.

The simulation, when using an external controller, should use the *RealtimeSimulatorImpl*. However, because the simulation time on a scenario with more than 4 H-SDNs is slower than in the real time, the default *SimulatorImplementationType* is used. The delay from the controller is therefore not constant, since it depends on the number of events processed by the NS-3 in the time that the controller takes to process the message and respond to it. However, a delay can be added to the packets entering the *tap* bridge, if needed.

4.2 Wireless OpenFlow Channel - LTE

Southbound interfaces were designed with Ethernet in mind. However, for mobile nodes, the OpenFlow channel needs to be wireless. Adapting such interface to wireless links comes with challenges, such as the increasing time between request and reply, more frequent connection loss, lack of coverage, etc..

Taking into consideration all the wireless technologies provided, LTE was chosen due to the ratio between covered area and bandwidth for more demanding services like video transmission and others. In a real scenario, the controller would be in a remote location, accessible through the Internet. In NS-3, this description is emulated by a remote host connected to a network that has access to the LTE Core. Moreover, Figure 8 shows an example of this connection to two hybrid SDN devices, where one eNodeB provides coverage to the entire scenario. This is done by the function `InstallExternalController` shown in Figure 3, where one eNodeB is installed and a CSMA channel is created between the PGW node and the controller to simulate the Internet. This CSMA interface in the controller is then used by the *tap* bridge module to connect to the External Controller.

Afterwards, in the function `CreateOpenFlowChannels`, each H-SDN is attached to the eNodeB. However, because the Internet backbone is not configured, and is being simulated by a CSMA link, the ARP tables in the remote host need to be updated manually with the H-SDN LTE interfaces' IP and the MAC of the PGW. This is done by the function `PopulateARPCacheWithSDNinterfaces`, and only then is the OpenFlow channel configured.

Afterwards, the Ryu controller is connected to the remote host node by a *tap* bridge interface, and the respective routes are added in the operating system, so the RYU controller knows where to send the OpenFlow packets.

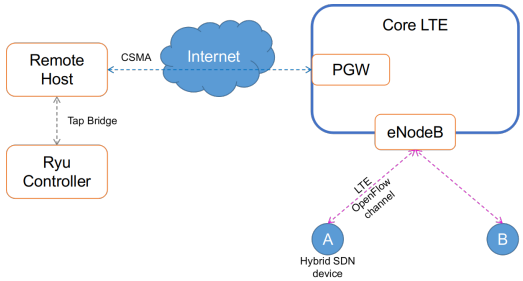

Figure 8: Simulation environment with an Out-of-band OpenFlow channel (from each H-SDN device to the Remote Host) connected to an NS-3-External Controller.

5 ROUTING APPLICATION AND TOPOLOGY DISCOVERY PROCESS

The routing is performed like in a normal SDN routing application, by sending a *Packet_Out* message with the actions for the received *Packet_In* message, or sending an *OpenFlow Modification* message with the rule for a specific matched field.

In the proposed platform two distinct handlers are used to parse a packet: the ARP and IP handlers. They use the topology graph (*networkx* library Graph) to search for a path to the destination using the Dijkstra algorithm [13]. If no path is found, the packet is sent to the legacy upper layers. However, in order for the routing application to work, the controller needs an accurate and up-to-date view of the entire network, provided by the topology discovery mechanisms. If such mechanisms do not work properly, this will affect the routing logic and decreases the performance [1]. The frequency of such updates is a trade-off between network overhead and accuracy. For scenarios with less movement, the updates can be less frequent.

SDN controllers use OpenFlow Discovery Protocol (OFDP), which encapsulate Link Layer Discovery Protocol (LLDP) frames in an OpenFlow packet-out message. Nevertheless, because the interfaces are wireless, the topology discovery process cannot be the

same used for Ethernet links, and adapting this protocol to wireless environments has several limitations [5]:

- The assumption that links are point-to-point, which is not true in wireless networks;
- A wireless link does not have a binary state (on/off) like wired links;
- LLDP protocol is not prepared to provide node or link related attributes.

Therefore, we propose the following topology discovery mechanism that takes advantage of *Packet_In* messages (like in a normal SDN routing application), and the Experimenter messages sent by each H-SDN.

Figure 9 shows how the topology discovery mechanism works when receiving a *Packet_In* message. When a *Packet_In* message is received, Node 1 can update the *datapath_id* and *port_no* from the OpenFlow header. Moreover, the *isOF* flag can be set to true because the sender's node is an H-SDN node. Afterwards, depending on the type of technology, the MAC header is parsed, which in a wireless link, indicates that the Node 2 is directly linked to Node 1. However, because it is a wireless link, Node 2's IPv4 address may not be the same address carried by the packet.

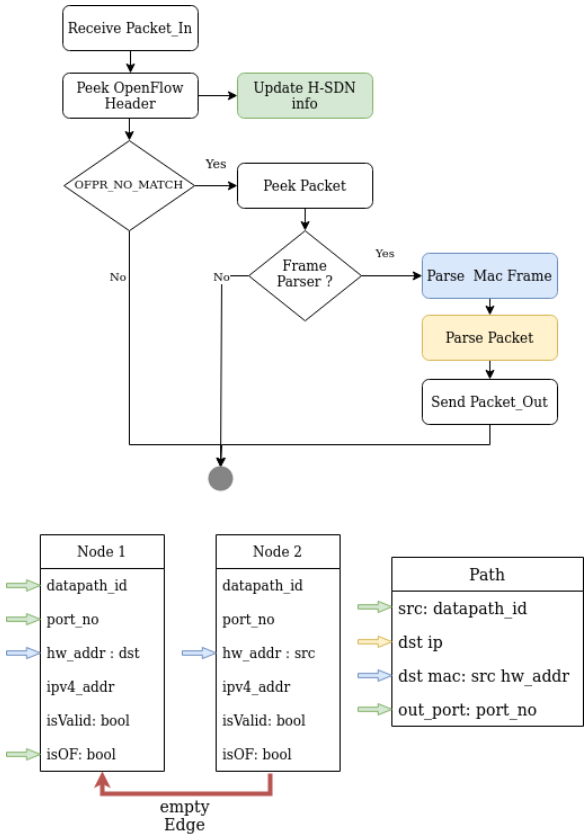

Figure 9: Topology discovery mechanism: *Packet_In*

When the IP Handler is used and a direct packet is received (like for example an *OLSR* Hello message), we can conclude that Node 2's *ipv4_addr* is the source IP address. However, when other IP packets

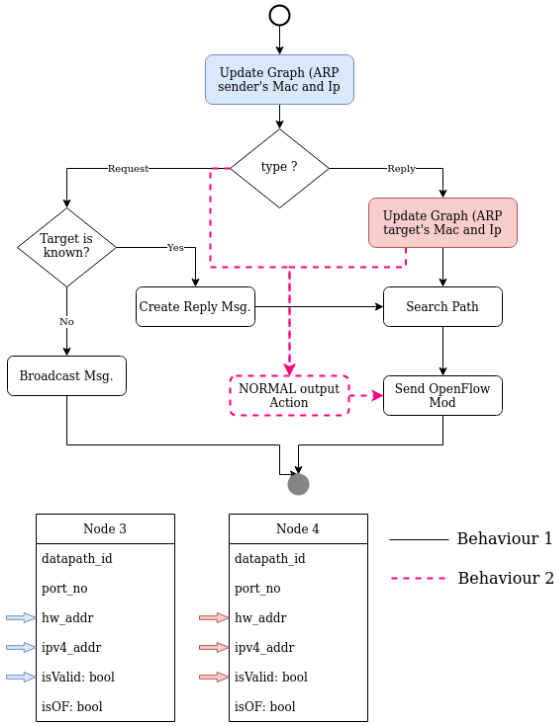

Figure 10: ARP handler behaviour.

are received (IP Handler), nothing can be updated in Node 2. We can only infer that two nodes exist, one with the source IP address (Node 3), and the other with the destination IP address (Node 4).

When the ARP Handler is executed, two behaviours can happen, as shown in Figure 10. The behaviour 1 (in black) is based on the ARP Handler used by RYU, in which the controller replies to the ARP request. The behaviour 2 (in pink) sends the ARP messages to the legacy upper layers, so the local ARP table can be constructed. The second behaviour is advantageous for networks with a faulty controller connection because it takes less time to fill the local ARP table in case the H-SDN node behaves like a legacy device. Despite the two behaviours, the topology is acquired in the same way for wireless interfaces. In case of an ARP request message, Node 3 is created with the senders' MAC and IP fields. For an ARP reply message, Node 4 is created with the target's MAC and IP fields.

Moreover, a path can be created when considering Node 1 and Node 2 from Figure 9 in cooperation with: (1) Node 3 created by the ARP Handler if the message is an ARP request; or (2) Node 4 in case of an ARP reply; or (3) Node 3 created by the IP Handler. This new path describes a route from Node 1 to Node 3/4, where Node 2 is the next hop of the packet. However, in this new path, no metric can be inferred about the links (besides the fact that they exist).

In order to prevent conflicting rules between H-SDN and legacy devices, if a packet received in a H-SDN device has a rule in which the next device matches the sender of the packet, a new rule is added, where the legacy device is avoided. This new rule has a small expiration time, so the device is not avoided even when the network changes.

5.1 OpenFlow Experimenter message

Due to the assumption that the legacy network will work on *ad-hoc* mode with a legacy routing protocol, we use the updated local routing tables to gather a list of the 1-hop neighbours of the H-SDN in the network. This process is done by the monitoring tool, by merging the information gathered through the received packets, the updated routing tables, and other device's info, into the message described in Figure 11.

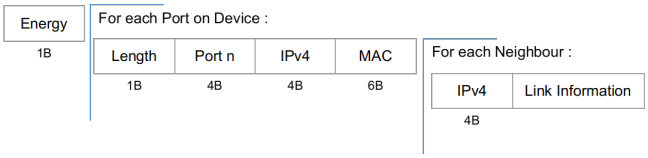

Figure 11: OpenFlow Experimenter message payload.

Afterwards, the controller uses Algorithm 1 to process these messages. First, we extract the node's energy; then, for each SDN port, we parse the IP and MAC addresses that compose *NodeA*. Then, for each neighbour of that OpenFlow port, we extract the neighbour's IP address (which describes *NodeB*), and the link information, which is then used to add the edge between these two nodes with the link information. Moreover, between ports, we also add an edge with value zero, meaning that they are in the same node (leaving space for further development). With the link information on each edge, the routing algorithm can perform a better path search based on metrics.

Algorithm 1: Experimenter Message Parser Algorithm

```
Node_old ← Null;
X ← msg;
N ← len(msg);
UpdateEnergy(X);
while N ≠ 0 do
    l ← Length;
    port, ip, mac ← GetVals(X);
    NodeA ← GetNode(port, ip, mac);
    while l ≠ 0 do
        ;                          /* For Each Port on the Device */
        vals ← Get_link_information(X);
        ip ← GetIp(X); NodeB ← GetNode(ip);
        AddEdge(NodeA, NodeB, vals);
        l ← l − len(vals, NodeB);
    end
    if Node_old not Null then
        AddEdge(NodeA, Node_old, 0);
        ;                                        /* Same device */
    end
    Node_old ← NodeA;
    N ← N − Length;
end
```

Furthermore, every H-SDN device, before sending any *Packet_In* message, first sends an *OpenFlow Experimenter* message, so the controller knows the MAC and IP addresses (*hw_addr* and *ipv4_addr* respectively) associated with the SDN port (*port_no*) to that device (*datapath_id*).

However, this Hybrid SDN architecture is only advantageous when the majority of paths do not have multiple hops between legacy devices. The topology discovery mechanism will not be able to quantify those links. It may be aware that the nodes exist, but not their connections, which in time may affect the network performance, or even create loops because the majority of the routes would not be under the controller's rules.

6 CASE STUDY SCENARIO

To study the performance of the proposed simulation environment, a scenario was prepared where every device starts in a position given by the *GridPositionAllocator*, with a step size of 50 meters, and a *GridWidth* of 5 columns.

The mobility pattern is based on the *RandomWaypointMobilityModel* with a Position Allocator given by the *RandomRectanglePositionAllocator* of NS-3, where the next position is chosen randomly inside a rectangle with the same size as the grid. Afterwards, each node moves towards that position at a constant velocity (*Speed*= $3m/s$), without any *Pause*.

Each device is equipped with an IEEE 802.11ax NIC with a Modulation Code Scheme (MCS) index of 3, a frequency of 5 Ghz, a channel width of 40 Mhz, and a guard interval of 800 ns.

As the routing legacy protocol, OLSR is used because it is one of the most common protocols for *ad-hoc* environments.

All results shown in the next sub-sections are averages over five or ten executions of each simulation with a standard deviation.

6.1 Environment Performance

We compare the scenario regarding two performance metrics, the effective simulation time (using the system clock), and memory usage (using the Valgrind tool [25]), on a computer with 16GB RAM and AMD Opteron(tm) Processor 6128. Regarding the traffic pattern, every device creates an UDP flow with every other device, creating flows for all the combinations (of two devices) in that scenario. For example, 5 devices create $C(5,2) = 10$ flows. In the results, the network size ranges between a total of 8 nodes (4 legacy and 4 H-SDN devices) to 16 nodes (4 legacy and 12 H-SDN devices, or 12 legacy and 4 H-SDN devices). The process repeats until the end of the simulation that lasts 30 seconds.

Figure 12 shows the results for the effective simulation time (the real time that the simulation takes), where each curve represents a specific data rate. In the first three curves, the experiment maintains the number of legacy devices (4), and the last curve represents an increase of the number of legacy devices with 4 H-SDNs. By adding one H-SDN device, the effective simulation time increases, on average, by 200 seconds for the 100 kbits data rate, and 160 seconds for the 1 Mbits. Comparatively, when only increasing the number of legacy devices at 100 kbits, with 16 nodes (4 H-SDN devices), we get a run time of 1335 seconds, which is 33% lower than with 12 H-SDN and 4 legacy devices.

Figure 13 shows the results of memory usage, where each curve represents different data rates for the UDP flows. As expected, the memory usage increases when higher data rates are used. The biggest contributor for the increase of memory usage when increasing the number of nodes is the NS-3 application, while RYU only spends on average 20 MB.

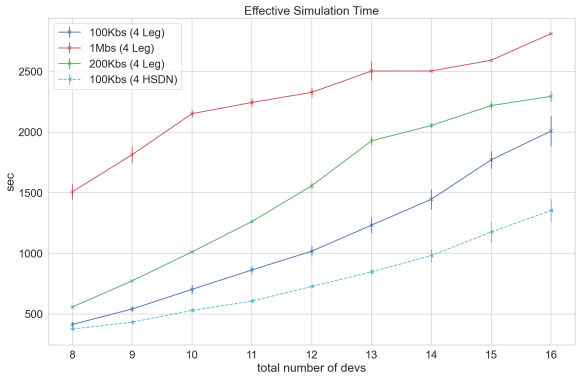

Figure 12: Effective simulation time for the case study scenario.

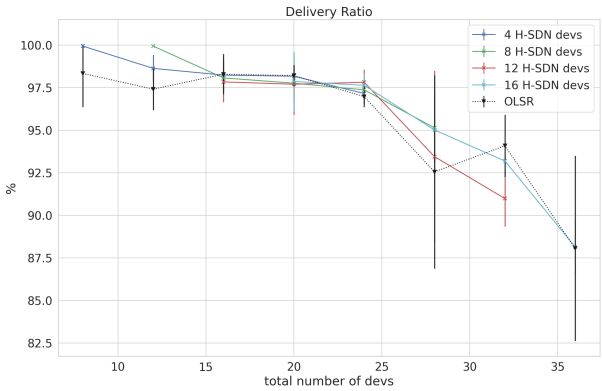

Figure 14: Packet delivery ratio.

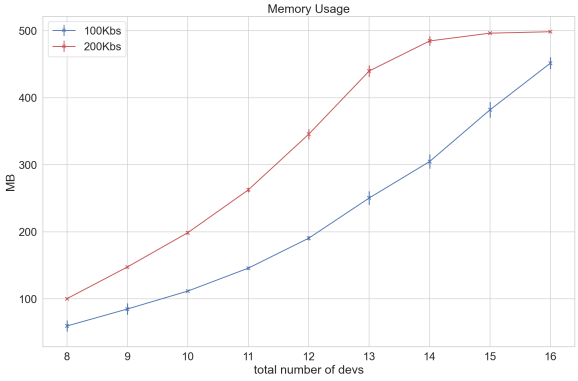

Figure 13: Memory usage for the case study scenario.

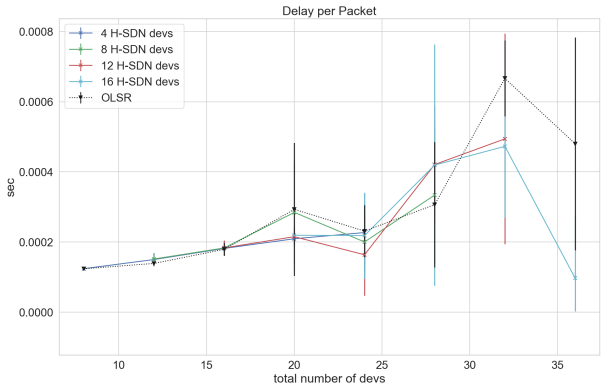

Figure 15: Mean delay.

6.2 Network Performance

To test our first hypothesis (whether a detach control plane can benefit MANETs), a test case was prepared in the same scenario described before, where the proposed architecture is compared to an only-OLSR MANET. However, the number of flows is fixed to 40 UDP flows, where the senders (chosen from the first grid column) and receivers (chosen from the last grid column) are randomly selected. Each flow has a data rate of 100 Kbits with a 500 bits packet size.

Figures 14 and 15 show the results, where we test different numbers of legacy and H-SDN devices. The X axis represents the total number of devices in the experiment, where the number of legacy devices is the total number of devices minus the number of H-SDN devices used in each curve, whose number is given by the legend of the figure. The black curve represents an exclusive legacy network.

As shown in Figure 14, this scenario shows that the proposed architecture has a higher delivery ratio for some points when compared to an all legacy network. Furthermore, when comparing the same amount of nodes but in different curves, we observe that, on average, the delivery ratio increases when the number of H-SDN is higher. This can be due to the centralized view of the network gathered by the controller, which in turn updates the OpenFlow rules faster than the spread of new neighboring information with the OLSR "Hello" messages. Despite the number of flows being the same throughout the experiment, we observe that, with an increase in the number of network nodes, the delivery ratio decreases. This can be explained by the increasing probability of a randomly selected pair having more hops in the path when a higher number of nodes are allocated. With a higher number of hops comes a higher probability of collision drop or propagation loss, therefore decreasing the delivery ratio.

As expected, with a higher number of total devices, meaning a higher number of hops in each route, the delay tends to increase, as shown in Figure 15. In addition, when comparing the OLSR curve with the other curves, for some points the delay can be lower when using H-SDN devices. This is due to the up-to-date view in the controller, which describes a better shortest path when comparing to an older network view on some legacy devices. Furthermore, the search path mechanism uses the shortest path metric, not contemplating any key performance indicators besides the number of hops. Therefore, when adding other routing mechanisms to the controller, a larger increase in the performance can be achieved. We can also state that, depending on the scenario in hands, a proper configuration of the SDN environment, *i.e.* applying the right set of rules, may lead to a performance increase in the MANET.

7 CONCLUSIONS AND FUTURE WORK

This paper proposed an approach for NS-3 using a WiFi mobile ad-hoc network, with legacy and hybrid SDN-capable devices, a cellular OpenFlow channel, and an external controller, interconnecting legacy devices to an SDN network. The performance results show that the novel NS-3 environment can be scaled to support large networks. However, the proposed H-SDN device increases the effective simulation time. The test case results show that this architecture can have at least the same performance as an OLSR-only solution. However, if another mechanism with QoS parameters other than only the number of hops are used, a greater performance can be achieved.

This approach will be further improved with more elaborate routing mechanisms, and will be used as the platform for the test of emergency scenarios and their algorithms.

ACKNOWLEDGMENTS

This work was partially supported by the "Fundação para a Ciência e Tecnologia" (FCT) through the Grant SFRH/BD/143516/2019, and Project 813391 TeamUp5G, in the framework of the Marie Skłodowska-Curie Innovative Training Networks.

REFERENCES

[1] Abdelhadi Azzouni, Nguyen Thi Mai Trang, Raouf Boutaba, and Guy Pujolle. 2017. Limitations of openflow topology discovery protocol. In *2017 16th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*. 1–3. https://doi.org/10.1109/MedHocNet.2017.8001642

[2] P. Bellavista, A. Corradi, and C. Giannelli. 2010. The real Ad-hoc Multi-hop Peer-to-peer (RAMP) middleware: An easy-to-use support for spontaneous networking. In *The IEEE symposium on Computers and Communications*. 463–470. https://doi.org/10.1109/ISCC.2010.5546785

[3] Daniel J Casey and Barry E Mullins. 2015. SDN shim: Controlling legacy devices. In *2015 IEEE 40th Conference on Local Computer Networks (LCN)*. IEEE, 169–172.

[4] M. Chan, C. Chen, J. Huang, T. Kuo, L. Yen, and C. Tseng. 2014. OpenNet: A simulator for software-defined wireless local area network. In *2014 IEEE Wireless Communications and Networking Conference (WCNC)*. 3332–3336. https://doi.org/10.1109/WCNC.2014.6953088

[5] Lunde Chen, F Toulouse, Pascal Berthou, and Kokouvi Bénoît Nougnanke. 2017. A Generic and Configurable Topology Discovery Service for Software Defined Wireless Multi-Hop Network. (2017), 101–104.

[6] Samir R Das, Charles E Perkins, and Elizabeth M Royer. 2000. Performance comparison of two on-demand routing protocols for ad hoc networks. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, Vol. 1. IEEE, 3–12.

[7] Chaves et. al. 2016. OFSwitch13: Enhancing Ns-3 with OpenFlow 1.3 Support. In *Proceedings of the Workshop on Ns-3* (Seattle, WA, USA) *(WNS3 '16)*. Association for Computing Machinery, New York, NY, USA, 33–40. https://doi.org/10.1145/2915371.2915381

[8] Gupta et. al. 2015. NS-3-based real-time emulation of LTE testbed using LabVIEW platform for software defined networking (SDN) in CROWD. In *Proceedings of the ACM Workshop on NS3, Barcelona, May 2015.*

[9] Jaime Galán-Jiménez. 2018. Exploiting the control power of SDN during the transition from IP to SDN networks. *International Journal of Communication Systems* 31, 5 (2018), e3504.

[10] J. F. Gonzalez Orrego and J. P. Urrea Duque. 2017. Throughput and delay evaluation framework integrating SDN and IEEE 802.11s WMN. In *2017 IEEE 9th Latin-American Conference on Communications (LATINCOM)*. 1–6. https://doi.org/10.1109/LATINCOM.2017.8240186

[11] Saad H Haji, SR Zeebaree, Rezgar Hasan Saeed, Siddeeq Y Ameen, Hanan M Shukur, Naaman Omar, Mohammed AM Sadeeq, Zainab Salih Ageed, Ibrahim Mahmood Ibrahim, and Hajar Maseeh Yasin. 2021. Comparison of software defined networking with traditional networking. *Asian Journal of Research in Computer Science* (2021), 1–18.

[12] Teerawat Issariyakul and Ekram Hossain. 2009. Introduction to network simulator 2 (NS2). In *Introduction to network simulator NS2*. Springer, 1–18.

[13] Adeel Javaid. 2013. Understanding Dijkstra's algorithm. *Available at SSRN 2340905* (2013).

[14] Wonyong Yoon Junhyuk Park, Janfizza Bukhari. 2019. An SDN Testbed for Evaluating Wireless Heterogeneous Networks. In *Journal of Theoretical and Applied Information Technology*, Vol. 97.

[15] Dimitrios Kafetzis, Spyridon Vassilaras, Georgios Vardoulias, and Iordanis Koutsopoulos. 2022. Software-Defined Networking Meets Software-Defined Radio in Mobile ad hoc Networks: State of the Art and Future Directions. *IEEE Access* 10 (2022), 9989–10014. https://doi.org/10.1109/ACCESS.2022.3144072

[16] Karamjeet Kaur, Japinder Singh, and Navtej Singh Ghumman. 2014. Mininet as software defined networking testing platform. In *International Conference on Communication, Computing & Systems (ICCCS)*. 139–42.

[17] Sukhveer Kaur, Japinder Singh, and Navtej Ghumman. 2014. Network Programmability Using POX Controller. https://doi.org/10.13140/RG.2.1.1950.6961

[18] Atta Ur Rehman Khan, Sajjad A Madani, Khizar Hayat, and Samee Ullah Khan. 2012. Clustering-based power-controlled routing for mobile wireless sensor networks. *International journal of communication systems* 25, 4 (2012), 529–542.

[19] Ian Ku, You Lu, Mario Gerla, Rafael L Gomes, Francesco Ongaro, and Eduardo Cerqueira. 2014. Towards software-defined VANET: Architecture and services. In *2014 13th annual Mediterranean ad hoc networking workshop (MED-HOC-NET)*. IEEE, 103–110.

[20] Mohamed Labraoui, Charalampos Chatzinakis, Michael Mathias Boc, and Anne Fladenmuller. 2016. On addressing mobility issues in wireless mesh networks using software-defined networking. In *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE, 903–908.

[21] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. 1–6.

[22] Dan Levin, Marco Canini, Stefan Schmid, Fabian Schaffert, and Anja Feldmann. 2014. Panopticon: Reaping the {Benefits} of Incremental {SDN} Deployment in Enterprise Networks. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. 333–345.

[23] Akshatha Nayak Manjeshwar, Arghyadip Roy, Pranav Jha, and Abhay Karandikar. 2019. Control and Management of Multiple RATs in Wireless Networks: An SDN Approach. In *2019 IEEE 2nd 5G World Forum (5GWF)*. 596–601. https://doi.org/10.1109/5GWF.2019.8911703

[24] Tesnim Mekki, Issam Jabri, Abderrezak Rachedi, and Lamia Chaari. 2021. Software-defined networking in vehicular networks: A survey. *Transactions on Emerging Telecommunications Technologies* (2021), e4265.

[25] Nicholas Nethercote and Julian Seward. 2003. Valgrind: A program supervision framework. *Electronic notes in theoretical computer science* 89, 2 (2003), 44–66.

[26] nsnam. 2008. OpenFlow switch support. Retrieved "May 2022" from https://www.nsnam.org/docs/models/html/openflow-switch.html

[27] Erick Petersen and Marco Antonio To. 2020. DockSDN: A Hybrid Container-Based SDN Emulation Tool. In *2020 IEEE Latin-American Conference on Communications (LATINCOM)*. 1–6. https://doi.org/10.1109/LATINCOM50620.2020.9282297

[28] Khalid Ibrahim Qureshi, Lei Wang, Liang Sun, Chunsheng Zhu, and Lei Shu. 2020. A Review on Design and Implementation of Software-Defined WLANs. *IEEE Systems Journal* 14, 2 (2020), 2601–2614. https://doi.org/10.1109/JSYST.2019.2960400

[29] Michael Rademacher, Florian Siebertz, Moritz Schlebusch, and Karl Jonas. 2016. Experiments with OpenFlow and IEEE802.11 Point-to-Point Links in a WMN. *ICWMC 2016 - Twelfth Int. Conf. Wirel. Mob. Commun.* November (2016), 99–105.

[30] George F Riley and Thomas R Henderson. 2010. The ns-3 network simulator. In *Modeling and tools for network simulation*. Springer, 15–34.

[31] Sachin Sharma, Avishek Nag, Paul Stynes, and Maziar Nekovee. 2019. Automatic Configuration of OpenFlow in Wireless Mobile Ad hoc Networks. *2019 International Conference on High Performance Computing and Simulation, HPCS 2019* (2019), 367–373. https://doi.org/10.1109/HPCS48598.2019.9188200

[32] Nithin Shastry and Keerthan Kumar T G. 2020. *Enhancing the Performance of Software-Defined Wireless Mesh Network*. 1–14. https://doi.org/10.1007/978-981-15-2612-1_1

[33] Klement Streit, Nils Rodday, Florian Steuber, Corinna Schmitt, and Gabi Dreo Rodosek. 2019. Wireless SDN for Highly Utilized MANETs. In *2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. 226–234. https://doi.org/10.1109/WiMOB.2019.8923172

[34] S. Wang. 2014. Comparison of SDN OpenFlow network simulator and emulators: EstiNet vs. Mininet. In *2014 IEEE Symposium on Computers and Communications (ISCC)*. 1–6. https://doi.org/10.1109/ISCC.2014.6912609

[35] Shie-Yuan Wang, Chih-Liang Chou, and Chun-Ming Yang. 2013. EstiNet openflow network simulator and emulator. *IEEE Communications Magazine* 51, 9 (2013), 110–117.

[36] Saleh Yousefi, Mahmoud Siadat Mousavi, and Mahmood Fathy. 2006. Vehicular ad hoc networks (VANETs): challenges and perspectives. In *2006 6th international conference on ITS telecommunications*. IEEE, 761–766.

[37] Liehuang Zhu, Md M Karim, Kashif Sharif, Chang Xu, Fan Li, Xiaojiang Du, and Mohsen Guizani. 2020. SDN controllers: A comprehensive analysis and performance evaluation study. *ACM Computing Surveys (CSUR)* 53, 6 (2020), 1–40.