Integration of WAVE and OpenFlow for Realization of SDN-based VANETs in ns-3

Juan Leon, Yacoub Hanna, and Kemal Akkaya
Department of Electrical and Computer Engineering
Florida International University
Miami, Florida, USA
{jleon148,yhann002,kakkaya}@fiu.edu

ABSTRACT

While software defined networking (SDN) has been widely deployed in wired networks within the cloud data centers, its applications with both a wireless control and data channel have not been fully realized. One particular use case is vehicular ad hoc networks (VANET), where SDN has been touted as an effective method to control vehicles' data traffic in a centralized manner. Several studies explored how SDN switches can be utilized on connected vehicles to shape the data traffic by enabling multi-hop communications through their Dedicated Short Range Communication (DSRC) interfaces. However, evaluation of research for SDN-based VANETs became a challenge due to lack of testbeds and realistic simulation tools that will support VANET features in conjunction with SDN. In this paper, we present a new framework that will enable SDN-based VANET simulation in ns-3. Specifically, we demonstrate how ns-3 SDN module can be expanded to support DSRC interfaces in addition to their default CSMA-CD interface. By considering a platooning application use case, we implemented an SDN-based VANET simulation that will enable multi-hop communication through the DSRC interfaces of the vehicles and road side units (RSUs). We not only explain the details of the implementation within the ns-3 OpenFlow (OF13) switch module but also provide some sample experiment results under a variety of conditions. The results show that the implementation is robust and closely simulates the actual behavior of SDN's control and data channels.

CCS CONCEPTS

• **Networks** → **Network simulations**; **Network measurement**; **Network mobility**; *Ad hoc networks.*

KEYWORDS

SDN, OpenFlow, OFSwitch, WAVE, DSRC, VANETs, Platooning

ACM Reference Format:
Juan Leon, Yacoub Hanna, and Kemal Akkaya. 2022. Integration of WAVE and OpenFlow for Realization of SDN-based VANETs in ns-3. In *Proceedings of the WNS3 2022 (WNS3 2022), June 22–23, 2022, Virtual Event, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3532577.3532608

WNS3 2022, June 22–23, 2022, Virtual Event, USA

ACM ISBN 978-1-4503-9651-6/22/06…$15.00
https://doi.org/10.1145/3532577.3532608

1 INTRODUCTION

Software Defined Networking (SDN) is the separation of data and control planes on switches for efficient management of data traffic and offering flexibility against dynamic conditions through a centralized controller [3]. The concept has also been effective in dealing with interoperability challenges that may arise due to vendor specific switch technologies. Since its inception more than a decade ago, SDN technologies became very disruptive and has started to be used in many modern IT infrastructures, particularly within the data centers. This is because SDN offer a robust control channel which can provide fast and reliable access to every switch and those switches might need to route data around to offer load balancing of the traffic. As such, most of the SDN technologies focused on Ethernet-based switches and development of controllers that will follow a standard for controller to switch communications (i.e., Open Flow standard [8]).

However, the implications of SDN is not limited to data centers and wired/optical networking. There has been many research on adapting SDN capabilities for many other applications where either the control channel or one of the data channels would be based on wireless technologies [2],[23]. In particular, with the maturity of 5G technologies, the control channel of SDN can now be based on 5G connections since it can offer the level of stability and bandwidth that exists for wired networks. Thanks to such developments, SDN concept can now be realized in wireless data planes to deal with the dynamicity and mobility of the switches within their application context.

A perfect example is vehicular ad hoc networks (VANETs) [13] which considers networking of vehicles (i.e., vehicle-to-vehicle) and road side units (RSUs) (i.e., vehicle-to-infrastructure) based on their wireless interfaces specifically designed for safety purposes. DSRC [11] was introduced to broadcast vehicle location and speed to its neighboring vehicles to eliminate potential traffic accidents. Similarly, this information can also be used by RSUs to offer information about the traffic and road conditions to other vehicles in the vicinity. DSRC was the protocol based on IEEE 802.11p and supplemented by higher layer protocols such as Wireless Access for Vehicular Environments (WAVE). SDN was introduced for VANETs to make more flexible use of vehicle's data forwarding capabilities by enabling an on-board unit that can support an SDN switch. In particular, this is meaningful when there is large number of devices that need to be controlled in response to certain phenomena (i.e., traffic route updates). The topic has been widely studied under different use cases to explore how SDN's features can be best utilized.

Nevertheless, when it comes to evaluation of SDN-based VANETs, there has been many struggles. There are multiple factors adding to

this challenge. First of all, there are not many tools supporting the simulation of SDN for wireless environments. Mininet [10], which is the main emulator for SDN-based research assumes a wired controller channel through CSMA-CD-based links and supports only addition of WiFi interface to the SDN switches. Therefore, utilization of Mininet for VANET-like applications was not possible. As a result, majority of the published work on SDN-based VANET either does not offer a realistic evaluation or demonstrates a small-scale proof-of-concept using IoT devices such as Raspberry PIs. While ns-3 had included SDN capabilities through its Open Flow module, this was mainly designed to work with CSMA-CD or WiFi not other wireless standards like WAVE. This largely hindered the feasibility of large-scale accurate evaluation of SDN-based VANETs.

Therefore, in this paper, we present our design and implementation for enabling the integration of SDN and WAVE to support the realization of SDN-based VANETs in ns-3. The main contribution is to enable the marriage of OpenFlow module with WAVE network devices so that an SDN switch that sits on a vehicle will be able to communicate with the SDN controller, makes rule updates on the rule table and forward data through the WAVE interface if needed. We also implement a 4G-based control channel and build a switch that can support multiple interfaces such as WAVE, CSMA-CD and others. To the best of our knowledge, this is the first realistic implementation of SDN-based VANETs in ns-3.

To demonstrate a proof-of-concept for our ns-3 extension, we utilized a platooning application for VANETs where two platoon leaders try to re-connect when they move away from each other. Since there is no long-distance communication, we propose utilizing SDN and enable multi-hop routing from one leader to the other through the intermediary RSUs. Through ns-3, we simulated a variety of scenarios with different number of vehicles and RSUs with realistic channels of 4G and WAVE supported by ns-3. We present results that show the level of overhead of using SDN and how the performance is impacted at scale compared to alternative approaches.

The rest of the paper is organized as follows. In the next section, we summarize the related work that considered SDN-based simulation efforts. Section 3 provides some background on the topic. In Section 4, we detail our design and implementation in ns-3. Section 5 presents some performance results under a platooning use case. Finally, we conclude the paper in Section 6.

2 RELATED WORK

There are several prior studies related to SDN-based VANETs [9][7], [12][20]. While some of these studies focused on routing among vehicles using SDN, others used SDN to access RSUs. But in any case, none of these studies provided an accurate and comprehensive evaluation setup to demonstrate a proof of concept implementation.

Only a few papers in the literature have constructed and used a simulation environment or an actual testbed to demonstrate that their approach can allow a real evaluation beyond theoretical debates and models such as [14]. In [18], Using Raspberry Pis as Openflow switches, the authors create a realistic implementation that was SDN-based for WiFi integration into VANETs. They show how SDN makes resource management in networks with restricted bandwidth available. Their drawback is that their testbed could only handle Wi-Fi, ignoring SDN's widespread application for long-distance management over wired networks. In [5], the authors were showing performance advantages in managing the communication as well as networking services and by expanding the use of Mininet emulator to illustrate SDN applications in SDN-based VANETs. On the other hand, the controller-to-vehicle link is still presumed to be wired, which is impossible in the real world. Our work introduces 4G-based wireless control for VANETs in ns-3, which is now the only practical and realistic option for remotely controlling automobiles and allows the integration of WAVE with SDN switches.

Having a realistic simulation of VANETs are required to analyze emerging technologies based on these networks and to demonstrate the advantages derived from their evaluation [22]. The two most important aspects of VANET simulation are network communication as well as vehicle mobility. Since the network architecture changes dynamically as the vehicles travel, these two properties are related to each other. Therefore, there are several useful simulators that can run VANETs environments. However, none of them can capture the realities of VANET environment like ns-3. In addition, ns-3 simulation for VANETs is more scalable and easier to use, with the major benefit being ongoing support and quick expansion due to the huge developer community. ns-3 allows for establishing the Ad hoc wireless networks as well as implementing several other protocols. As such ns-3 is the most adaptable and solid tool used in the VANET simulation platforms [19][6]. It is important to note that more features suited for VANETs are being implemented in ns-3, such as advances in channel and devices models as well as the implementation of vehicle mobility models.

3 BACKGROUND AND PRELIMINARIES

3.1 SDN and OpenFlow

The SDN architecture contains the SDN Controller and the network switches as shown in Figure 1. Through the OpenFlow protocol, the SDN Controller sets the rules and the logic for the SDN switches in order to handle the packet forwarding. The SDN controller has a comprehensive overview of the switch network (in our case vehicles and RSUs) and can compute the routes from host to host.

Switches in the SDN-OpenFlow architecture provide a standard remote controller interface called OpenFlow [17]. The controller can use this protocol to find OpenFlow compatible switches, establish matching rules for switching hardware, and gather information from switching devices. The Controller may create, edit, and delete the flow entries in the flow tables using the OpenFlow re/pro-actively. In each switch, there is a flow table that consists of several flow records which are the matching fields, counters as well as rules to apply the matched packets. Furthermore, if a switch receives a new packet that is not matching the flow entries, it delivers it as a packet to the controller using the command *packet in*.

3.2 OFSWITCH13 Module

ns-3 already supports OpenFlow 1.3 (the latest version of OpenFlow) in a module called `OFSWITCH13 Module` [4]. This module provides a controller application interface as well as the switch device to the ns-3 simulator. Therefore, by using this module it is feasible to link several ns-3 nodes to transmit and receive traffic

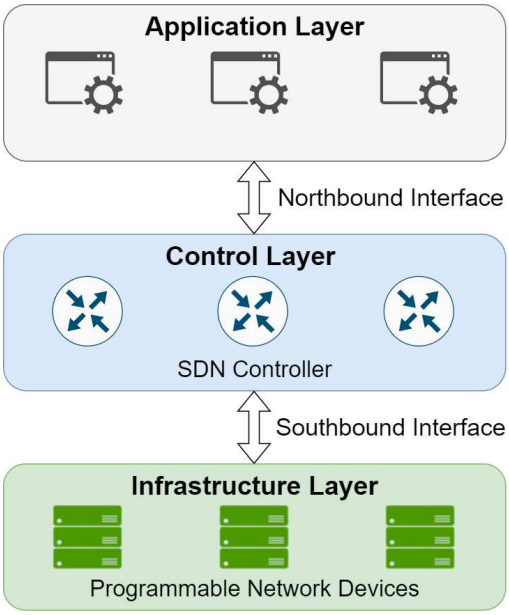

Figure 1: SDN Infrastructure

using the current Ethernet (i.e., CSMA-CD) devices. In order to implement a network using `OFSWITCH13`, the controller application interface could be expanded to execute any needed control logic in which the communication between the switch and the controller is based on the standards of ns-3 in terms of devices, channels as well as a protocol stack. Moreover, it is worth mentioning that this module depends on external libraries to be compatible with ns-3. This library enables the use of switch data path which includes configuring the switch using the data paths command line (DPCTL) utility tool as well as converting OpenFlow messages that are coming back and forth from a wire format.

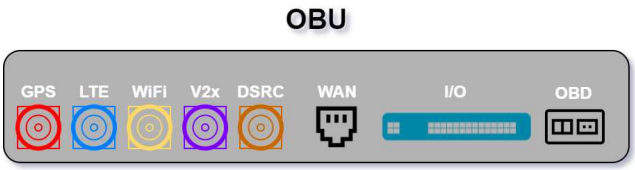

Figure 2: Sample OBU Interfaces [21]

3.3 WAVE Protocol Stack (IEEE 1609)

The Wireless Access in Vehicular Networks (WAVE) protocol was developed to offer an interoperable communication interface for the transmission of V2I as well as V2V messages. The architecture, security mechanisms, and the message format are all defined by the protocol stack. IEEE 802.11p which is the modified version of 802.11, is used for the physical channel access. Therefore, as previously stated, it is also known as DSRC. This standardization allows for a wide range of applications, including traffic control and safety.

The On-board Unit (OBU) and RSU are the major components that utilize this protocol stack. As illustrated in Figure 2 every OBU on a vehicle may be equipped with several interfaces such as GPS, WiFi, DSRC, LTE, CSMA-CD, etc. connections. These are considered as the standard interface features for an OBU. We envision that an SDN switch will be connected to the OBU through its CSMA-CD interfaces while it connects to SDN controller on a separate LTE/4G/5G interface. In this way, the DSRC interface will be used for V2V and V2I (Vehicle to RSU) communications and CSMA-CD will be used only to send/receive packets from the application layer running on the OBU to the switch. Note that DSRC devices transmit within a range that can go up to 1000 meters, and with a data transmission rate around 27 Mbps [16]. Both Wave Short Message Protocol (WSMP) and IP protocol are supported on top of the MAC layer, as illustrated in Figure 3. However, there is no current support for multi-hop communication among WAVE devices in the standard. The current ns-3 simulator supports use of CSMA-CD devices but does not allow integration of other interfaces limiting its use to a wired environment only.

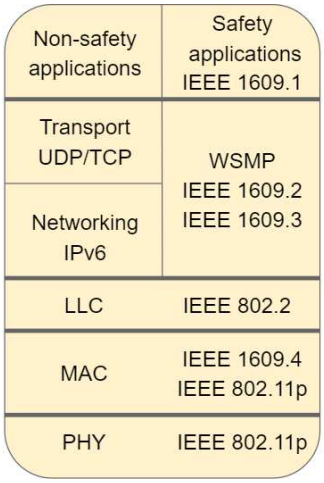

Figure 3: WAVE Protocol Stack

4 NS-3 EXTENSIONS FOR INTEGRATION OF WAVE AND SDN

The integration of WAVE module with SDN required a series of changes in the ns-3 codes, particularly **OFSWITCH13** [4] module which supports Open Flow. In the following subsection, we explain these changes and additions.

4.1 Motivation and Overview

The `OFSWITCH13` module permits different end devices to be connected to a single SDN switch allowing them to communicate with each other. To achieve this, each switch associates the end devices with a corresponding port number. Then, packets can be forwarded by the switches to this corresponding port. Currently, the only supported NetDevice to be connected to a switch is a `CSMANETDEVICE`.

As mentioned before, in VANETs vehicles use a specialized communication interface called DSRC which represents the Physical layer and MAC layer (IEEE 802.11p) of the WAVE protocol stack. In ns-3, the WAVE module provides access to NetDevices capable of

utilizing the IEEE 1609 WAVE protocol stack and the underlying physical DSRC devices. Therefore, there is a need to integrate the `OFSWITCH13` module with the WAVE module.

4.2 Initialization of the SDN Switch in ns-3

The `OFSWITCH13` module in ns-3 assumes that multiple end devices will be connected to an individual switch. This switch will then allow the end devices to communicate with each other by forwarding traffic from one port to the corresponding port.

However, our setup in VANET is a bit different. Specifically, we would like to create a VANET architecture where each SDN switch will only service one individual end device (i.e., a vehicle or RSU). This means that on each vehicle or RSU there will be an SDN switch installed with a direct CSMA-CD link to the vehicle's On-Board Unit (OBU). We will consider this to be PORT 2 for each SDN switch. Every SDN switch will also be connected to a DSRC capable device. In ns-3, the WAVE NetDevice has the WAVE protocol stack alongside the lower layer DSRC communication. We will consider this DSRC to be PORT 1 for each SDN switch. Note that PORT 1 and PORT 2 of the SDN switches are the ports that will constitute the data plane as shown in Figure 4.

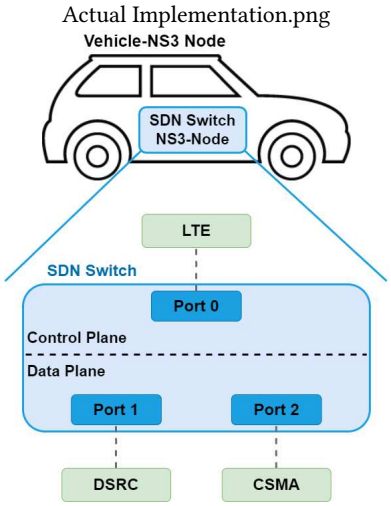

Figure 4: SDN Switch on Each Vehicle in ns-3

Each SDN switch will also be connected via the standard ns-3 `LTE` module to the SDN Controller. We will consider this to be PORT 0 for each SDN switch. It is important to note that in the case of the SDN controller there will be only one communication interface which is LTE in our case. PORT 0 of the SDN switches alongside PORT 0 of the SDN controller constitutes the Control plane.

The SDN switch will then be able to forward the packets from PORT 1 (OBU) to the desired interface. In order to determine the desired interface, each switch will receive OpenFlow table updates from the SDN controller. Due to having SDN switches on each vehicle as well as on each RSU, we are able to achieve control of V2V and V2I communications. In other words, the SDN switch will provide access to control the physical communication interfaces on the vehicle based on the logic defined by the SDN controller.

4.3 Integrating WAVE for OFSWITCH13 Module

In order to achieve the interoperability of the `WAVE` module with the `OFSWITCH13` module, certain modifications to the ns-3 OFSWITCH13 module were needed. Specifically, these changes were made to the `ofswitch13-device` and `ofswitch13-port` classes.

Binding the SDN Control Channel: In the `ofswitch13-device`, there was no available method to explicitly bind the NETDEVICE for the control channel with the socket that will establish the SDN controller to SDN switch communication. As seen in Figure 5, the class `ofswitch13-device` will have a modification to its `StartControllerConnection()` method. This modification constitutes of a call to the method `BindToNetDevice()` inherited from the `NS3::Socket` class. Here, the NETDEVICE that is used as control channel interface will be linked to a TCP socket connecting the controller to the switch. The variable `n-ctlrDev` is used to represent a positive integer which is equivalent to the PORT number used for the control channel. Assuming the port enumeration as described above, the value for `n-ctlrDev` would be 0.

Port Integration: The next modification we made was within the `ofswitch13-port` class. As seen in Figure 5, two methods of the `ofswitch13-port` class were modified. The first method to be modified is the `OFSwitch13Port::NotifyController Connection()` method. This method allows the SDN controller to be notified which ports were added/created. This is where we link the reception of packets at the NETDEVICE level with the higher layer port reception abstraction. We begin by linking the MAC layer address of our LTE NETDEVICE to the `hw-addr` variable of the `ofsoftswitch13` library. Since this is an external library, proper formatting of the MAC Address is required to match the needs of `OFSwitch13Port:: NotifyControllerConnection()` method.

Cross-layer Reception: We then proceed to link the higher layer `OFSwitch13Port::Receive` method to the lower layer receive methods of the NETDEVICES. To do this, each NETDEVICE, (see Figure 5) will need to add a new method called `SetOpenFlow ReceiveCallback()`. This method will allow the NETDEVICES in the lower layers to call higher layers receive method.

The other method to be modified is the `OFSwitch13Port::Send`. Similarly, there needs to be a link made between the `OFSwitch13Port::Send()` method and the *Send* method of the underlying NETDEVICE. To this end, if the underlying NETDEVICE that calls the `OFSwitch13Port::Send()` method is a WAVENETDEVICE, we proceed to call `WaveNetDevice::SendX()` method. This enables the SDN switch to send or receive packets through its ports without having to worry about the underlying network protocols.

4.4 Extending TLV OXM-MATCH

As we extend `OFSwitch13` module, there is also a need for customization of the possible HEADERS to match with Openflow table rules. To this end, OpenFlow's *Extended Match Type Length Value*

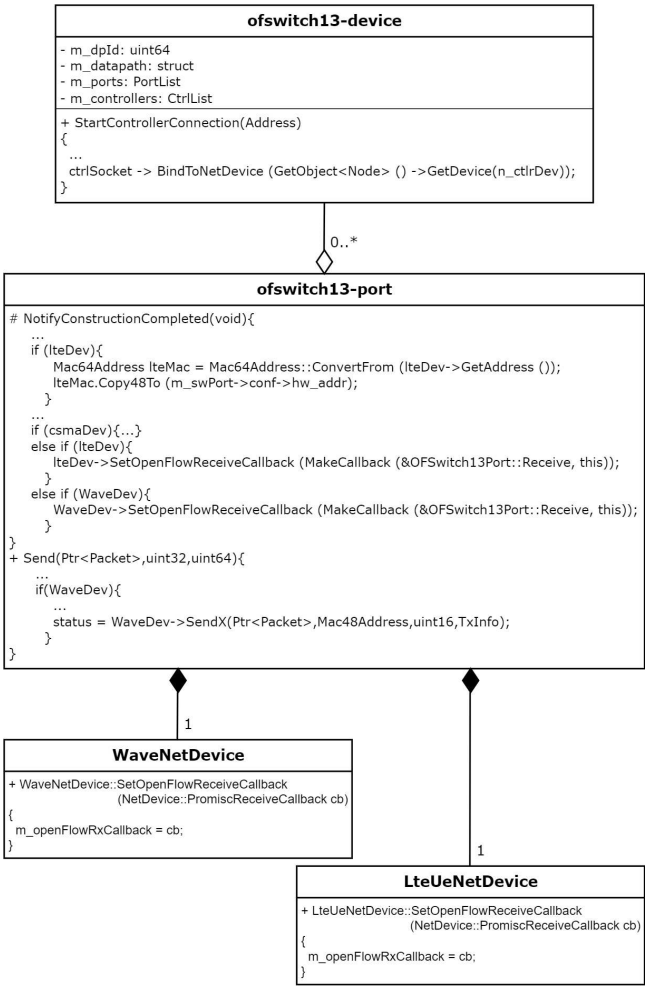

Figure 5: UML Diagram for the Extensions to OFSWITCH13 Module

(TLV) structure is used. There were two main challenges. First, there is little or outdated documentation regarding how to include new custom Match fields. This is because up to OFSWITCH13 version 3.3.0 the NetBee library was used to decode and parse packets. Starting from OFSWITCH13 version 4.0.0, the dependence on the NetBee library was removed. Second, there were no instructions on how to include a custom OXM Match structure. Having access to only the predefined fields limits the functionality of the module greatly. In this section, we detail the steps required to develop a custom OXM TLV structure with custom fields.

By following the steps performed in [4], we are left with a new custom OXM match field but with no way of utilizing the value contained in the fields. This is because the NetBee dependency that was used to parse the packets and insert/extract values into their OXM Match fields but it was removed.

Thus, two modifications are needed to be able to use a custom OXM TLV structure. The first is that the OXM match structure itself needs to be defined. This includes its type (i.e., *Protocol Number*) and which *Field* values it posses. Second, the size of the different

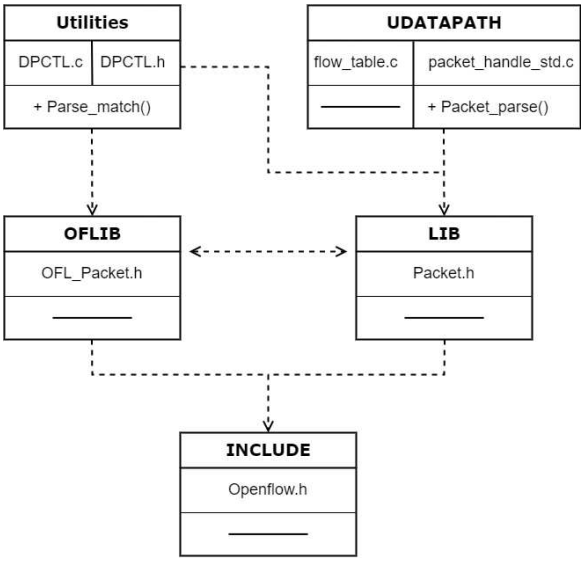

Figure 6: UML Diagram

fields that will be contained within that OXM TLV structure needs to be known and defined. This is because depending on the size and amount of fields, the total OXM TLV structure size will change. In order to do this, the management utility called *DPCTL* and its underlying dependencies needs to be updated to recognize its field along with inserting/extracting the new OXM match structure.

Creating OXM TLV Fields: The DPCTL management utility exists under the OFSOFTSWITCH 13 library in the UTILITIES directory. Within this directory, the class declaration `DPCTL.h` needs to modified in order to include a new OXM Match field. The Match field will allow the DPCTL utility to recognize what size does the custom match field has and consequently, the set of methods to properly insert values to the match field. Here is where the name of the custom field to be matched should be defined. Next, `DPCTL.c` class definition needs to be modified. From here, the method `parse_match()` needs to be modified to be able to recognize and then extract its actual content to parse it into a format acceptable by our custom OXM Match field. To this end, the methods `ofl_structs_match_ put16()`, `ofl_structs_match_ put32()` and `ofl_structs_match_put64()` may be used depending on the size of the variable used to contain our custom field. We next need to create our custom OXM Match structure to contain our OXM Match fields. This is done by modifying two classes within the UDATAPATH directory.

Creating OXM TLV Structure: The first class to be modified is the class declaration `packet_ handle_std.c`. Here, we modify the method `packet_parse()` to define our custom methods to process the OXM Match structure. Subsequently, in `flow_table .c` class, we create a new `oxm_id` with our custom OXM Match structure name. This allows our new OXM Match structure to be recognized by Openflow and consequently the set of methods required to access its OXM Match fields.

The next class to be modified is the class declaration `packet_handle_std.c` from the same directory. We modify the method `packet_parse()`. Here, we define the methods that will be used when we identify our OXM Match structure. This implies how to access the fields within the structure once the OXM Match type is matched. Next, the structure definition will take place within the LIB directory. Here, the class definition `packet.h` needs to be updated to include our definition of the custom OXM Match structure. The total size of the OXM match structure will be defined as well as its type. Also, the fields and their variable size need to be defined since they will affect the total structure size. In order for Openflow to recognize our OXM Match structure type, we also need to modify the class declaration `ofl_packet.h` from the directory OFLIB. This is done by linking the name of our custom structure type to a predefined constant value. Finally, in class declaration `openflow.h`, within the directory INCLUDE, we link our structure to the OXM TLV format in order for Openflow to recognize that the structure is an OXM Match structure.

5 PERFORMANCE EVALUATION

In this section, we present a series of experimental evaluations to show as a proof-of-concept of the implementation.

5.1 Simulation Setup and Assumptions

To demonstrate the use of the `OFSWITCH13` module with wireless technologies we devised a series of experiments utilizing the WAVE module alongside the LTE module. The experiments are based on a platooning application commonly used in the VANET environment. The scenario we consider consists of two platoons as shown in Figure 7. We assume that each platoon will be composed of a single vehicle chosen as Platoon leader followed by series of vehicles that will act as platoon followers. We assume the leaders are equipped with an SDN enabled switch alongside an LTE module to enable the control channel. The data channel will be composed of DSRC communications as well as wired CSMA-CD communication from the SDN switch to the OBU. Both platoons will be randomly placed within a defined area. One platoon (e.g., Platoon A) will be selected to attempt to merge the other platoon (e.g., Platoon B). If we assume a platoon management protocol as the one defined in [1], in order for platoons to merge they require to exchange constantly information regarding their velocity, acceleration and position with the platoon they are attempting to merge with. The *Beacon Safety Message* (BSM) is used to convey this information in VANET environments. Yet, due to the design of the WAVE protocol stack, the BSM cannot use multi-hop capabilities to travel. To be able to enable long range platoon coordination we take advantage of the OFSWITCH13 module to enable the SDN capabilities within ns-3 allowing multi-hop capabilities for the BSM. We achieve this by taking into consideration the MAC addresses of the vehicles and RSUs in relation to their actual position. Then, we can use any route calculation algorithm such as Dijkstra's routing algorithm to find the best route and not introduce any unnecessary overhead to the network. We also make the code available through Github [1]

[1]https://github.com/JVoltagic/NS3-ADWISE.git

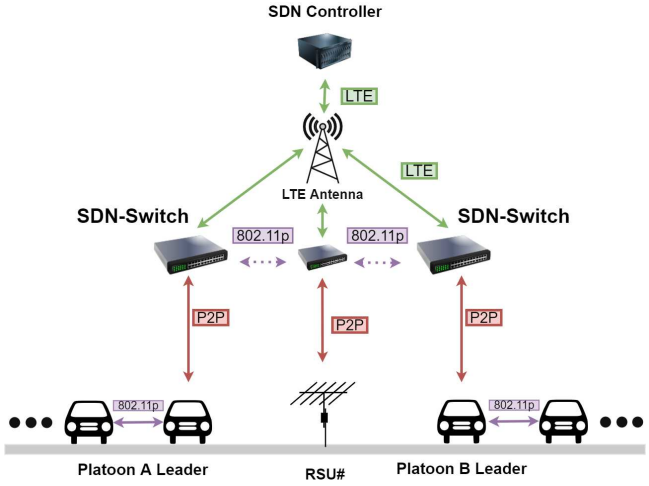

Figure 7: Platoon Communication through RSUs

5.2 Performance Metrics and Bench marking

We used the following metrics to measure the performance of our approach.

- *Setup Delay:* This measurement is used to show how long it takes the SDN controller to contact each individual switch to establish a connection. The time taken to send the proper rules is also accounted for in this metric.
- *Network Overhead:* The use of broadcast packets among RSUs adds to the network's overhead. This metric is measured by the number of extra messages that was generated within the network.
- *Average Packet Delay:* This statistic quantifies the time it takes for a packet to reach at its target vehicle (i.e., the leading platoon leader) from the point of origin. Since CACC communications older than 100ms are deemed outdated, the delay is an important parameter.

To compare with an alternative approach, we considered using Ad-hoc On Demand Distance Vector (AODV) [15] routing which is a distributed solution to address the connectivity among the two platoon leaders. Basically, the vehicles floods the network to reach out to everyone and eventually find the best path for packets to take to the lost platoon leader. We will use the same topology for the both approaches to ensure significant results.

5.3 Simulation Results

In this section, we provide the results obtained from ns-3 simulations for the metrics defined in various network environment settings. In each experiment, we run the simulations with either the SDN Controller or the AODV protocol handling the routing of BSM packets.

5.3.1 ***Distance and RSU Density Effects on Setup Delay:*** To evaluate the performance of our approach we observed the setup delay of the SDN approach with a series of experiments. The first experiment consists of increasing the distance between the controller and the SDN enabled switches. In this experiment we utilized 50 RSUs and 12 vehicles. As shown in Table 1, even when the distance between our controller and switches was greater than 15 km the

setup delay was not affected. We attribute this result to the low latency offered by the LTE module which indicates that realization of SDN for real-time applications is possible and effective.

Table 1: Distance Vs Setup Delay for our SDN Controller

Distance	Setup Delay
1km	8.127s
5km	8.127s
10km	8.127s
20km	8.127s

The next experiment consisted on changing the total amount of RSUs available. In other words, changing the amount of nodes that will receive rule updates from the SDN controller. Since we want to observe the effects on the Setup Delay we perform this experiment in a 250x250 meter size field with a total of 12 vehicles producing noise. We first try a low density of RSU in order to observe the effects of progressively increasing it until the density of RSUs is enough to saturate the field area. From Figure 8 we can observe a considerable difference in the SDN setup delay as we increase the number of RSUs. The difference stems from the nature of SDN communications. The more nodes that require rules the bigger the setup delay will be because the SDN controller will need to contact each additional node to establish a connection and only then can he start transmitting the rules. Yet, one important observation is the fact that the setup delay will increase linearly in relation to the node count even in extreme high node count scenarios.

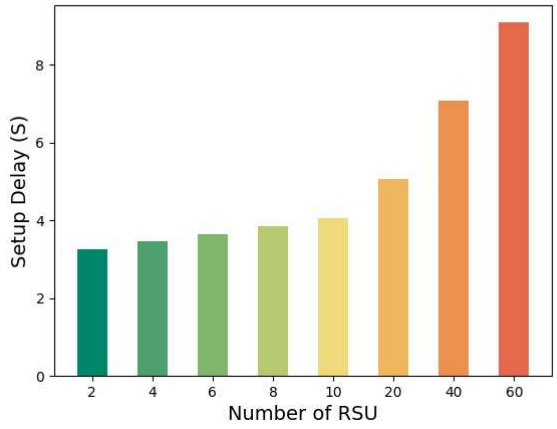

Figure 8: Setup Delay for SDN

5.3.2 ***Scalability Experiments:*** In these evaluations, we explore how the number of vehicles affects the performance of the SDN-based network. When comparing the vehicles noise (increasing the number of vehicles) we will compare the performance of our approach against the AODV benchmark. We expect to see the same benefits from using traditional SDN technologies but now in conjunction with the WAVE module. The vehicle count has been modified from 20, 40, 60, 80, and 100. Our experiments are performed in a 250x250 meter field with a total of 50 RSUs.

Network Overhead: When we increase the number of vehicles producing background traffic, as shown in Figure 9, the overhead for AODV increases exponentially. In the case of the SDN approach, there is an increase but its growth is linear. We determined that this behavior occurs because our SDN approach can deal with the background traffic very easily by filtering at the MAC layer the sending address of the packet. If the packet is not supposed to transit through that RSU or vehicle, it gets dropped. Meanwhile, for AODV, overhead increases because it relies on finding routes on the fly, and it may take many extra messages for AODV to understand that a specific node may not have any available paths to send the message or that it is not part of the most optimal route.

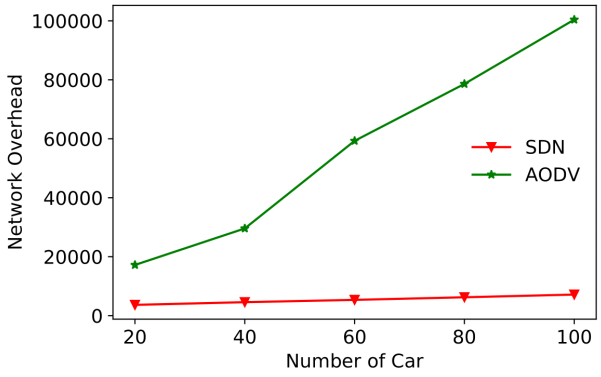

Figure 9: Network Overhead

Average Packet Delay: We can observe in Figure 10 an upward trend in the delay for AODV as the amount of background traffic grows with the increasing number of vehicles. This issue can be attributed to the need for AODV to find routes on the fly while contending with background traffic. In comparison to our SDN approach, where the rules are previously installed, we can easily distinguish between background noise and an actual BSM since the SDN switches only need to inspect the packet's MAC addresses to know if it is supposed to be dropped or re-transmitted.

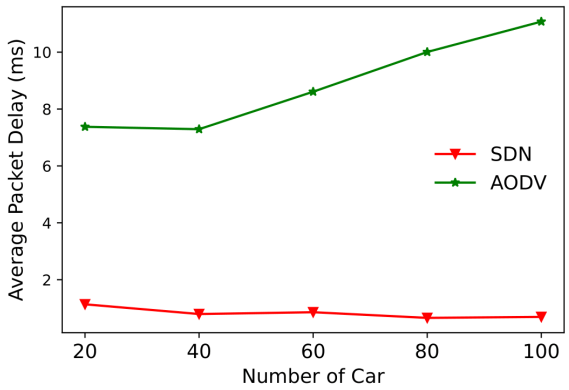

Figure 10: Average Packet Delay

6 CONCLUSION

In this paper, we introduced extensions to ns-3 OFSWITCH13 Module to support an enriched set of functions for the SDN switches. Specifically, we introduced how WAVE module can be integrated with OFSWITCH13 module to support for an SDN based VANET architecture. We also extended the possible OXM TLV structures that are available for the module in order to be able to match the new header fields that introduced by utilizing the WAVE protocol stack. The steps detailed here can also serve as guide to anyone interested in developing custom headers or implementing new modules in conjunction with the OFSWITCH13 module. By leveraging SDN technology in the VANET environment we are enabling the uncoupling of the data and control plane in order to enable multi hop capabilities for DRSC (WAVE protocol stack) communications. We explained in details how this implementation was done and demonstrated its feasibility in a use-case scenario based on vehicle platooning application. Through this application, we showed that our approach will support multi-hop routing for enabling efficient communication among platoon leaders which is much more effective than an existing benchmark solution such as AODV. As future works, we propose developing the OXM TLV structures required to handle additional fields and header structures required to follow the specifications of the WAVE protocol stack. This in turn will enable further research on the available applications in the VANET environment and their performance.

ACKNOWLEDGMENTS

This work is supported by US National Science Foundation under the Award Number CNS-1816112.

REFERENCES

[1] Mani Amoozadeh, Hui Deng, Chen-Nee Chuah, Michael Zhang, and Dipak Ghosal. 2015. Platoon Management with Cooperative Adaptive Cruise Control Enabled by VANET. *Vehicular Communications* 2, 2 (2015), 110–123. https://doi.org/10.1016/j.vehcom.2015.03.004

[2] Alcardo Alex Barakabitze, Arslan Ahmad, Rashid Mijumbi, and Andrew Hines. 2020. 5G Network Slicing using SDN and NFV: A Survey of Taxonomy, Architectures and Future Challenges. *Computer Networks* 167 (2020). https://doi.org/10.1016/j.comnet.2019.106984

[3] Kamal Benzekki, Abdeslam El Fergougui, and Abdelbaki Elbelrhiti Elalaoui. 2016. Software-defined Networking (SDN): a Survey. *Security and Communication Networks* 9 (2016). Issue 18. https://doi.org/10.1002/sec.1737

[4] Luciano Jerez Chaves, Islene Calciolari Garcia, and Edmundo Roberto Mauro Madeira. 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

[5] Ramon Dos Reis Fontes, Claudia Campolo, Christian Esteve Rothenberg, and Antonella Molinaro. 2017. From Theory to Experimental Evaluation: Resource Management in Software-defined Vehicular Networks. *IEEE Access* 5 (2017), 3069–3076.

[6] Agata Grzybek, Marcin Seredynski, Grégoire Danoy, and Pascal Bouvry. 2012. Aspects and Trends in Realistic VANET Simulations. In *2012 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. 1–6. https://doi.org/10.1109/WoWMoM.2012.6263793

[7] Zongjian He, Jiannong Cao, and Xuefeng Liu. 2016. SDVN: Enabling Rapid Network Innovation for Heterogeneous Vehicular Communication. *IEEE network* 30, 4 (2016), 10–15.

[8] Fei Hu, Qi Hao, and Ke Bao. 2014. A Survey on Software-defined Network and OpenFlow: From Concept to Implementation. *IEEE Communications Surveys and Tutorials* 16 (2014). Issue 4. https://doi.org/10.1109/COMST.2014.2326417

[9] Xiang Ji, HuiQun Yu, GuiSheng Fan, and WenHao Fu. 2016. SDGR: An SDN-based Geographic Routing Protocol for VANET. In *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 276–281.

[10] 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.

[11] John Kenney. 2011. Dedicated Short-Range Communications (DSRC) Standards in the United States. *Proc. IEEE* 99, 7 (July 2011), 1162–1182. https://doi.org/10.1109/JPROC.2011.2132790

[12] Ian Ku, You Lu, Mario Gerla, Rafael 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.

[13] Sandeep Kugali and Sneha Kadadevar. 2020. Vehicular Adhoc Network (VANET):-A Brief Knowledge. *International Journal of Engineering and Technical Research* 9 (06 2020). https://doi.org/10.17577/IJERTV9IS060784

[14] Juan Leon, Oscar Bautista, Abdullah Aydeger, Suat Mercan, and Kemal Akkaya. 2021. A General and Practical Framework for Realization of SDN-based Vehicular Networks. In *2021 IEEE International Performance, Computing, and Communications Conference (IPCCC)*. 1–7. https://doi.org/10.1109/IPCCC51483.2021.9679400

[15] Charles Perkins and Elizabeth Royer. 1999. Ad-hoc On-demand Distance Vector Routing. In *Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications*. 90–100. https://doi.org/10.1109/MCSA.1999.749281

[16] Mukesh Saini, Abdulhameed Alelaiwi, and Abdulmotaleb El Saddik. 2015. How Close Are We to Realizing a Pragmatic VANET Solution? A Meta-Survey. *ACM Comput. Surv.* 48, 2, Article 29 (nov 2015), 40 pages. https://doi.org/10.1145/2817552

[17] Sibylle Schaller and Dave Hood. 2017. Software Defined Networking Architecture Standardization. *Computer Standards & Interfaces* 54 (2017), 197–202.

[18] Gokhan Secinti, Berk Canberk, Trung Duong, and Lei Shu. 2017. Software Defined Architecture for VANET: A Testbed Implementation with Wireless Access Management. *IEEE Communications Magazine* 55, 7 (2017), 135–141.

[19] Evjola Spaho, Leonard Barolli, Gjergji Mino, Fatos Xhafa, and Vladi Kolici. 2011. VANET Simulators: A Survey on Mobility and Routing Protocols. In *2011 International Conference on Broadband and Wireless Computing, Communication and Applications*. 1–10. https://doi.org/10.1109/BWCCA.2011.11

[20] Kalupahana Liyanage Kushan Sudheera, Maode Ma, Nawaz Ali, and Peter Han Joo Chong. 2016. Delay Efficient Software Defined Networking Based Architecture for Vehicular Networks. In *2016 IEEE International Conference on Communication Systems (ICCS)*. IEEE, 1–6.

[21] Norbert Varga, László Bokor, András Takács, József Kovács, and László Virág. 2017. An Architecture Proposal for V2X Communication-centric Traffic Light Controller Systems. In *2017 15th International Conference on ITS Telecommunications (ITST)*. IEEE, 1–7.

[22] Julia Silva Weber, Miguel Neves, and Tiago Ferreto. 2021. VANET Simulators: an Updated Review. *Journal of the Brazilian Computer Society* 27, 1 (2021), 1–31.

[23] Yang Yang and Kun Hua. 2019. Emerging Technologies for 5G-Enabled Vehicular Networks. *IEEE Access* 7 (2019). https://doi.org/10.1109/ACCESS.2019.2954466