On the Implementation of a Cross-Layer SDN Architecture for 802.11 MANETs

Ilias Syrigos
Gridnet SA
Volos, Greece
is@gridnet.gr

Ippokratis Koukoulis
Gridnet SA
Volos, Greece
ik@gridnet.gr

Apostolis Prassas
Gridnet SA
Volos, Greece
ap@gridnet.gr

Kostas Choumas
Gridnet SA
Volos, Greece
kc@gridnet.gr

Thanasis Korakis
University Of Thessaly
Volos, Greece
korakis@uth.gr

Abstract—**The adoption of the Software Defined Networking (SDN) paradigm is aggressively expanding from traditional datacenter networks to 5G and IoT deployments due to its flexibility in network management and routing. Mobile Ad-Hoc Networks (MANETs), with their unique characteristics and requirements stemming from the mobility and volatile wireless medium conditions, have not yet realized the benefits of the SDN approach. Adoption has been hampered by technical challenges and the contradiction between the centralized concept of SDN and the distributed one of traditional MANET routing schemes. In this paper, we present our implementation of a fully-fledged SDN framework, which attempts to bridge the gap by combining the benefits of both worlds. Our developed prototype integrates with 802.11 Ad-Hoc networks and offers a) fault tolerance for the SDN controller by recovering from failures with the (re)-election of a new controller, b) dynamic network topology discovery, and c) a cross-layer approach in routing based on 802.11 MAC layer statistics that more accurately characterize link quality and capacity. We validated the applicability and performance benefits of our method by evaluating our proposed SDN architecture in a proof-of-concept scenario.**

Index Terms—**SDN networks, MANET, resilience, cross-layer, Raft, 802.11**

I. Introduction

Software Defined Networking (SDN) is an emerging, evolutionary paradigm that provides dynamic and efficient network configuration and management by decoupling the control plane from the data plane. This allows for a holistic and centralized view of the network, upon which the SDN controller makes intelligent routing and forwarding decisions. It enables the execution of Quality of Service (QoS) policies, the deployment of new, complicated and sophisticated routing protocols, and the administration of heterogeneous switching equipment without the need for additional hardware or manual configuration. SDN's flexibility in traffic management and efficiency in network operation have encouraged both industry and academia to swiftly adopt it as the dominant networking paradigm. According to [1], the SDN market is anticipated to witness a substantial growth from USD 13.7 billion in 2020 to USD 32.7 billion by 2025, at a Compound Annual Growth Rate (CAGR) of 19% during this period.

However, its adoption in wireless networks and, more specifically, in Mobile Ad Hoc Networks (MANETs) is not prevalent due to a number of unresolved research challenges, despite extensive academic study. The majority of obstacles originate from the dynamic and volatile wireless environment caused by the unpredictable mobility, arrival, and departure of wireless nodes, as well as the inherent conflict between the centralized structure of SDN networks and the distributed nature of MANETs. The SDN controller, the central entity, is susceptible to failures or disconnections that render the entire network inoperable. In addition, the dynamic movement of network nodes makes establishing connectivity channels between them and the controller a difficult puzzle to solve. These challenges go beyond the realm of research and are manifested in the form of technical difficulties and trade-offs that must be addressed for delivering a usable and efficient solution.

Traditional MANET routing protocols, such as Optimized Link State Routing Protocol (OLSR) [2], are able to adapt to dynamic network environment changes, exhibiting an acceptable performance, by leveraging their distributed nature to discover neighboring nodes and forward network traffic. However, the benefits of such protocols are accompanied by higher network overhead due to message exchanges and lengthy convergence times, which hinder performance in low-capacity and sparse networks, as well as in scaled and complex mesh networks. Their primary limitation, though, is the lack of a global network perspective with data coming from the various layers of the network stack, as well as the inability to act on this data with the application of intelligent policies, due to their lack of programmability.

This paper presents and evaluates our design and implementation of a fully-fledged SDN framework for MANET networks, which attempts to bridge the gap between distributed MANET protocols and centralized SDN approaches by adopting their strengths and overcoming their constraints. Specifically, our contributions can be summed up as:

- Definition and development of an in-band SDN architecture in which the connection to the controller, part of the MANET, is also SDN-based.
- Provision of fault-tolerance and resilience in the event of controller failure or out-of-range movement, with automatic (re-)election of a new controller.
- Development of a distributed, OLSR-like topology discovery mechanism that permits dynamic network composition and quick reaction to network changes.
- Integration with the wireless 802.11 protocol via tunnel-

ing and the development of agents that extract MAC/PHY layer statistics from the wireless card driver.

- Implementation of an adaptive forwarding process that is aware of the wireless network's utilization.

This is, to the best of our knowledge, the first attempt to develop a fully operational SDN framework that addresses the majority of MANET challenges and may be regarded as a competitive alternative to routing protocols such as OLSR.

The remaining sections of this paper are structured as follows: The section II provides a summary of relevant earlier work on SDN approaches for MANETs. Section III describes in detail each of our contributions while presenting the overall architecture of our developed framework. In section IV, a proof-of-concept use case deployed on a wireless testbed and a mobility scenario deployed in an emulation environment are used to evaluate the framework. Section V, concludes the paper by reviewing our proposed scheme and findings and identifying future research directions.

II. Related Work

Originally, the SDN concept has emerged for providing flexibility and dynamic management for the centralized networks of datacenters and mobile networks' backhauls. Nevertheless, several approaches have explored the integration of SDN with wireless networks, particularly wireless mesh networks. Prior work in [3] provides a hybrid framework that employs OpenFlow to route data traffic, while it uses OLSR for routing OpenFlow control traffic and data traffic in the event of an SDN controller failure. The evaluation was conducted within an emulation environment. Additionally, the work in [4] represents a deployment that allows for automatic selection of an SDN controller through an election procedure that then imposes routing rules with a higher priority than those of the existing routing protocol (Babel) used for providing connectivity between the nodes.

The authors in [5], [6] propose a flexible SDN architecture in which nodes use backup paths discovered in a distributed manner, when there is detection of failure of the primary path installed by the SDN controller. Although this approach can provide quick reaction to network changes, it is not able to leverage a centralized network overview for performing optimal routing decisions. Another SDN-to-MANET integration effort is presented in [7] that exhibits quick reaction times to network volatility. However, the approach is based on the assumption that the nodes possess two wireless interfaces, with the one being used for the communication with the controller, which is assumed to be always in direct connection. The works in [8], [9] provide the foundation for QoS in SDN routing, the former by prioritizing traffic flows and the latter by considering the MANET's bandwidth utilization in taking routing decisions, although both are assuming a direct, always-available connection with a central entity responsible for managing the SDN network.

Our implementation, outlined in this paper, distinguishes itself from prior work in that, to the best of our knowledge, it is the first to offer a fully-fledged solution that addresses all aspects and issues of the integration of SDN into MANETs. It ensures deployment applicability by establishing and developing an in-band architecture requiring a single interface for both the control and data planes. In addition, the automatic election of the SDN controller improves robustness and eliminates single points of failure. Finally, and most importantly, it follows a cross-layer approach, incorporating MAC layer statistics for the accurate estimation of link capacities and the QoS routing of flows.

III. Proposed SDN Scheme

In this section we describe the entire SDN framework we designed and developed, providing an overview of the architecture along with details for each distinct process and functionality.

A. Overall Design

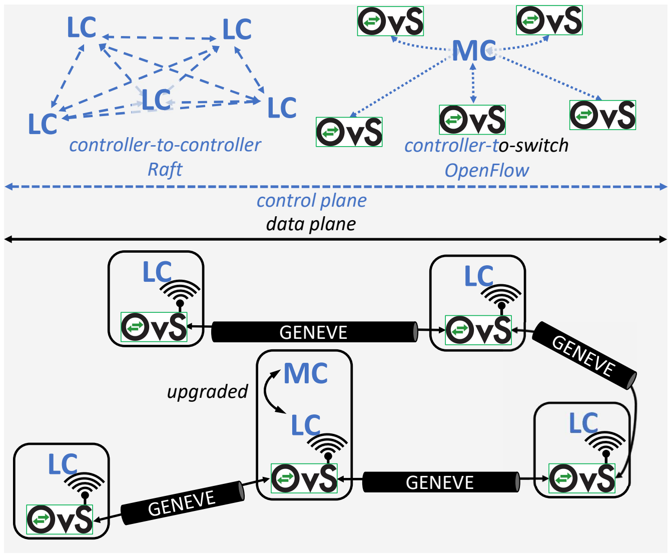

Fig. 1. Overall Architecture

A high-level description of our fault-tolerant SDN framework is depicted in Figure 1, where each node in the MANET network is running an SDN *Local Controller* (LC) application in parallel with a virtual switch, based in Open vSwitch (OVS) [10]. In general, in the SDN concept there are two options for the implementation of the control plane, which is the means of connection between the controller(s) and the switches. The first, the out-of-band control plane, connects the controller(s) to the switches via a separate network. The second, the in-band control plane, utilizes the same network for both control and data traffic. Although it is technically challenging, we opted for the second option as it offers us the ability to completely control the network in an SDN manner and it is applicable to devices with a single wireless interface. We provide more details on how we handle the SDN control plane establishment with the assistance of LCs and the routing of control messages in subsection III-E.

Furthermore, in order to ensure reliability and fault tolerance for our system, we have employed the Raft Consensus Algorithm [11] as the distributed protocol for the coordination of controller-to-controller communication between LCs and the election of a suitable *Master Controller* (MC), which is the node responsible for managing the SDN network of a MANET cluster and applying flow rules to the virtual switches, via OpenFlow, the de-facto controller-to-switch communication protocol. Ultimately, the entire SDN architecture resides on top of an 802.11 Ad-Hoc network, with which is seamlessly integrated to enable efficient control and intelligent routing.

B. Integration with IEEE 802.11

For the deployment of our SDN framework, the integration with the underlying wireless communication protocol was necessary. Although the framework is portable and directly applicable to any wireless infrastructure, we focused on 802.11 protocol (WiFi), as it is the prevalent in MANETs. In our setup, the wireless nodes are configured as Ad-Hoc (IBSS), capable of communicating directly with their one-hop neighbors. As described in [12], IEEE 802.11 uses a different addressing scheme than the traditional 802.3 standard, making integration with a virtual switch, specifically with OVS, more challenging. Currently, only IEEE 802.11 interfaces configured in 4-address mode can be directly connected to OVS. However, this mode cannot be configured when the wireless interface is in Ad-Hoc mode, as the third address of the MAC header is reserved for use as the BSSID address by the standard. To overcome this challenge, we used a strategy similar to that described in [13]. We employed GENEVE tunnels to create a virtual point-to-point link between two nodes and attached them as ports in OVS, thus making the use of IEEE 802.11 completely transparent for the virtual switch.

Besides interfacing with OVS, our cross-layer SDN architecture integrates with the 802.11 MAC layer for performing routing of traffic by leveraging MAC/PHY layer statistics that are exported by the drivers of the wireless cards such as the Packet Delivery Ratio (PDR), the TX PHY bitrate as well as the airtime utilization. The SDN controller is able, through our implemented extension of OVS and a standard OpenFlow procedure, to centrally collect them and gain an accurate estimation of the quality of the wireless links between nodes. Subsection III-F offer a more thorough analysis.

C. Fault Tolerance

The SDN Controller is the heart of the SDN network and the most critical component. As such, it is the network's single point of failure. In a MANET environment, where the Controller is mobile and moves out of the range of the switches/nodes, the network becomes dysfunctional, unable to establish new communication paths, adapt to changes in the wireless environment, or maintain centralized decision control in general. To address this challenge, we designed and developed a fault-tolerant SDN network in which each node can immediately assume the role of the MC when necessary. To do this, each node in the network, as mentioned earlier, runs an LC process concurrently with an OVS instance, which is configured to connect with all other LCs/nodes in the network. By exploiting OpenFlow's capability of assigning different responsibilities to controllers, we are able to avoid conflicts caused by concurrent and potentially contradictory flow entry installations in switches and coordinate the coexistence of numerous controllers appropriately. According to the OpenFlow specification 1.3.1 [14], a controller can operate in one of the following roles:

- Equal: This is the controller's default role. The controller has complete access to the switch and is equivalent to other controllers in the same role.
- Slave: A Slave controller has access to the switch in a read-only mode. The controller is prohibited from issuing controller-to-switch commands.
- Master: The Master controller has complete read/write access to the switch. At any given time, only one controller can be the Master.

Thus, while every switch in the network runs an LC process, only a single of them can be upgraded to the MC role at any given moment, able to control and apply flow rules to the rest, therefore retaining SDN's centralized nature. This controller is configured as the Master in the OvS instances of the network. The rest are configured as Slaves for all OVS instances apart the ones that are collocated, for which they are configured as Equals.

This strategy eliminates the single point of failure providing robustness and resilience to the network. In our implementation, the employment of Raft election scheme is exploited for the nomination of the MC by automatically declaring the single *leader*, in Raft terminology, of a cluster of nodes as the MC of the SDN network. Specifically, the controller of the Raft-elected leader node communicates its intention to become MC with the use of OpenFlow *Role Request* messages.

There are two parameters that are controlling the behavior of the Raft algorithm, and therefore the controller-to-controller communication of the SDN network: a) the *heartbeat interval* and b) the *election timeout*. Heartbeat interval (*hbeat_interval*) defines the frequency with which the leader will notify followers that it is still the leader. Election timeout (*election_timeout*) is how long a follower node will go without hearing a heartbeat before attempting to become leader itself. Both should be tuned appropriately so that leadership is transferred rapidly in case of leader failure or disconnection and at the same time unnecessary elections and extra overhead are avoided.

D. Topology Discovery

Every MANET protocol's core functionality is neighbor discovery, as any node must be aware of its one-hop neighbors in order to establish multi-hop connectivity with any other node in the network. This is no different in our SDN case. However, due to the centralized nature of the SDN network, the controller, and specifically in our implementation the MC must be aware of the entire network topology. A common procedure in SDN wired networks, but also in wireless [15], for topology discovery, is through centralized circulation of LLDP packets between the controller and the switches and the subsequent building of a network graph.

This centralized approach comes with two major disadvantages. The first is that an LLDP packet originating from an SDN controller may traverse the entire MANET network to arrive back to the controller as an OpenFlow *Packet In* message, for the discovery of just a single link. Accounting the fact that this process must be repeated for every node and every port of the node's virtual switch, periodically, it is evident that the overhead in the network grows significantly as the size of the MANET network grows. The second is that having only a single SDN controller aware of the network's topology (i.e. MC for our case), there is a risk of losing connectivity during elections or during bootstrapping of the system. In order to overcome these challenges, we implemented a more distributed approach, similar to the one of OLSR.

In specific, the collocated LC of each node, operating in Equal mode, instructs the switch to broadcast special messages through the broadcast GENEVE tunnel we use to avoid overhead of forwarding broadcast messages through individual tunnels. Such messages include the *Hello* messages that are used for discovering neighboring links, and *Topology Control* (TC) messages that are used to discover multi-hop paths, both encapsulated in an LLDP packet as separate sections. The Hello message section of the LLDP packet consists of the originator node identifier and a bit that indicates if the message has been rebroadcasted. The TC message section contains the originator node's incoming one-hop links which are learned from previous Hello messages received.

The LLDP packets containing the Hello and TC sections are broadcasted every *lldp_period* ms and are rebroadcasted by all nodes using sequence numbers in order to avoid loops. Every node has a local topology database where it stores the links that learns through LLDP exchanges. In case a new TC message arrives that it does not contain a previously stored link, then this link is deleted from the database. Accordingly, the record of a neighboring link in the database from where the node does not receive a Hello message within *timeout_period* seconds is also deleted. For non-neighboring links, if no TC message is received after timeout_period seconds from the last TC message received from the same originator, then all links stored in the database associated with this originator are deleted as well. Therefore, with this distributed approach, every node and its respective collocated LC is able to construct a network graph with the discovered nodes (neighboring or not) allowing network connectivity during network bootstrap or MC election.

E. Flow Entries Management

Our SDN framework must effectively handle a number of difficult special circumstances that arise throughout the operation of the network lifecycle. First and foremost is the management of the network bootstrap that will allow the discovery and connection between the nodes and the non-collocated LCs. As was already mentioned, the switch's collocated LC manages the switch in tandem with the MC, when operating in Equal mode. The LC uses the network graph learned from the topology discovery to setup flows in its collocated switch to route control traffic. These flows are also used to route data plane traffic when an MC has not yet been elected, or if the LC has not been connected (or lost its connection) to the elected MC, to service data traffic in the absence of an MC. As a result, even if they only handle their switch, the collocated LCs are able to create a network graph and coordinate the forwarding of packets, both control and data, to the discovered network using the discovery mechanism described in the preceding subsection. This process is carried out concurrently by all LCs during network startup, and once the absolute majority of the cluster has been discovered and connected, the election of an MC can be completed and the elected controller will then take control of managing data flows for the whole network.

The controller, either operating as an MC or collocated LC, instructs the switches to forward packets by installing flow rules based on the Dijkstra shortest path found on the graph that has built and annotated with appropriate weights. The process of calculating the weights will be analysed in the following subsection. Each flow rule installed has an *idle_timeout*, meaning that if the flow remains inactive for a period larger than the timeout, the flow is deleted and the controller is notified. This allows switches to inquire for new (perhaps better) paths, when the flow is activated again. Furthermore, the installed flow rules on the switches are also stored internally by the controller process, in order to be able to manage them in case the graph is altered by a network event such as a new link discovered or an existing link removed. In more detail, a separate thread of the process, periodically inspects the stored flow entries and queries the graph for better paths, and when one is found, new rules are installed on the involved switches via *Flow Mod* messages. This enables data traffic to be always forwarded through the links providing the best performance. Both idle_timeout and period of updating the flows are apt to judgement and configuration of the network administrator, as the associated performance benefits are dependent to the volatility and size of the network.

Furthermore, when an MC, during Packet In handling, is unable to retrieve a path towards a destination, then it removes the respective flow entry for this destination at the switch of the source node that generated the data packet (if it exists). This last condition refers to the case, when a packet leaves its source and arrives at an intermediate node, between source and destination, which inquires the MC via a Packet In message. In this case, where the MC is unable to forward the packet, the flow entry at the source should be removed, otherwise the packets will keep following a path with dead end.

F. Cross-layer SDN-MAC integration

A significant advantage that an SDN network can provide us over the use of traditional MANET routing protocols, is the ability of the SDN Controller to perform intelligent routing decisions based on the knowledge that collects and stores from across the network nodes and between the various layers of the network stack. To that end, we have enabled the integration between the SDN control plane and the 802.11 MAC layer, by implementing the mechanisms that allow the controller to acquire data and statistics from the open-source driver of the wireless card of each node, following a similar approach with our previous works [16], [17]. These statistics are leveraged by the controller in order to define the weights, for the links in the graph, which are accurately characterizing the links' available capacity. These weights are calculated every τ period using the equations below, where r is the median of the PHY rates reported by the driver for the same period.

In the CSMA/CA based MAC layer of 802.11 where multiple stations are competing for the access to the wireless medium, there are several factors that influence the actual throughput performance of a transmitting station and these are also related with the assesment of a link's available capacity. For a time interval $[t_0, t_0+\tau]$ the available capacity $AC(t_0, \tau)$ of a link between a source node s and the directly connected destination node d is estimated as follows

$$AC_{sd}(t_0, \tau) = \frac{1}{\tau}\int_{t_0}^{t_0+\tau} \mathrm{PDR}(t) \times (1-u(t)) \times C(r)dt. \quad (1)$$

From Equation 1 we can observe that the available capacity of a link for a specific time interval depends on the calculated Packet Delivery Ratio $\mathrm{PDR}(t)$, the fraction of time $u(t)$ during which the medium is sensed as busy (airtime utilization) and the maximum link capacity $C(r)$. It is worth to note here, that $u(t)$ refers not only to the proportion of time required for the transmission of the traffic of the network's nodes, but also accounts for the interference from external networks or devices. The $C(r)$ for an 802.11 link (n version and above), assuming no losses and a simplified model based on the analysis in [18], is in turn calculated as follows

$$C(r) = \frac{\mathrm{maxAMPDU}(r) \times L}{\mathrm{TxDelay}(r)}, \quad (2)$$

where $\mathrm{maxAMPDU}(r)$ is the maximum number of frames allowed into an AMPDU frame for a PHY rate r, and L refers to the UDP payload carried by a single IP packet. The TxDelay corresponds to the transmission delay of the AMPDU frame for the PHY rate in use, which also involves the T_{WIFI} delay caused by management and control frames along with the backoff delay, all associated with the standard process of an 802.11 transmission, and which equals $T_{\mathrm{WIFI}} = 249\mu s$, when CTS, RTS and ACK are transmitted with 6 Mbits/s PHY rate

$$\mathrm{TxDelay}(r) = T_{\mathrm{WIFI}} + T_{\mathrm{DATA}}(r), \quad (3)$$

where $T_{\mathrm{DATA}}(r)$ can be approximately defined as

$$T_{\mathrm{DATA}}(r) = T_{\mathrm{PH}} + \frac{\mathrm{maxAMPDU}(r) \times (S_{\mathrm{MAC}} + S)}{r}, \quad (4)$$

where $T_{\mathrm{PH}} = 20\mu s$ is the transmission delay of the PHY header, S_{MAC} is the size of the MAC header and S is the size of a single frame.

In order to avoid unnecessary complexity by introducing bi-directional edges in the graph, we decided to denote as the available capacity for a link, the minimum between those estimated for each of the link's end points. Therefore, we annotate each edge between two directly connected nodes, s and d in the graph, with a weight w_{sd}, calculated as

$$w_{sd} = \frac{1}{\min(AC_{sd}, AC_{ds})} \quad (5)$$

Consequently, when a Packet In for a new flow arrives, the controller performs shortest path computation in order to establish the forwarding path of the flow. However, during the process of periodically updating the flows, the controller inspects every link's available capacity in order to avoid congested links. In such links, where AC is under a pre-defined threshold thr, the controller rearranges the flows, starting with the one, f, with the less consumed bandwidth $bw_{f,sd}(t_0, \tau)$, in order to bring AC over the thr again. For each flow f and for the given time interval $[t_0, t_0+\tau]$, the consuming bandwidth for a specific link $s \to d$ is approximated by

$$bw_{f,sd}(t_0, \tau) = \frac{\mathrm{TxBytes}_f(t_0, \tau)}{r}, \quad (6)$$

where $\mathrm{TxBytes}_f(t_0, \tau)$ are the transmitted bytes for flow f as reported by the switch.

The aforementioned parameters for the estimation of links' available capacities and therefore the edges' weights, as well as consumed bandwidths, are polled and collected from the open-source driver of the wireless card by our implemented agent, which is an extension of OvS, responsible for communicating these statistics to the MC and to the collocated LC. The interval, over which the agent inquires the driver is called l2_sample_period and can be configured by the network administrator. Measuring PDR and airtime utilization for the specified interval is straightforward. The exchange of the statistics is still OpenFlow compatible as we are utilizing the OpenFlow *Experimenter* messages that are defined by the standard and which allow us to pass arbitrary data. Furthermore, the actual transmitted bytes for each flow that are reported by the switches are encapsulated in OpenFlow *Flow Stats* messages.

IV. Evaluation

In order to test and evaluate our developed framework, we deployed it on a realistic testbed environment and assessed its performance under a particular, but commonly encountered scenario. We focused on routing decisions in the presence of utilized links, which allowed us to further illustrate the rationale and performance benefits of our cross-layer SDN approach.

A. Testbed Setup

Our testbed comprises of six wireless nodes, which are essentially commercial off-the-shelf computers running Ubuntu 18.04 and equipped with an 802.11ac-capable wireless card. The card's chipset is QCA6174, and the corresponding open-source driver is ath10k. In addition, we have installed our framework, which consists of numerous software modules and scripts. We have specifically deployed our extended version of OVS, based on version 2.15.90, our expanded version of

TABLE I
EXPERIMENT PARAMETERS

Parameter	Value
lldp_period	2 s
timeout_period	20 s
hbeat_interval	100 ms
election_timeout	1000 ms
l2_sample_period	5 s

Ryu Controller [19], based on version 4.34, our developed SDN application that implements the controller architecture mentioned in section III, and etcd [20], as a distributed storage and implementation of Raft, version 3.6. Several scripts have been developed to automate the configuration and initialization of the wireless interface, the GENEVE tunnels, and the virtual switch. These scripts are executed as startup services.

As is obvious from the preceding section, there are a number of configurable system parameters that can considerably impact its performance. a) the lldp_period, b) the timeout_period, c) the hbeat_interval, d) the election_timeout and e) the l2_sample_period are the most notable. The values of these parameters are closely related to the characteristics of the network, such as its size and mobility of nodes, and they create a trade-off between rapid adaptation and network overhead. However, their optimal configuration is partially studied in our prior work [21], so we have specified the values that we deem most suitable for our case, which may be found in Table I.

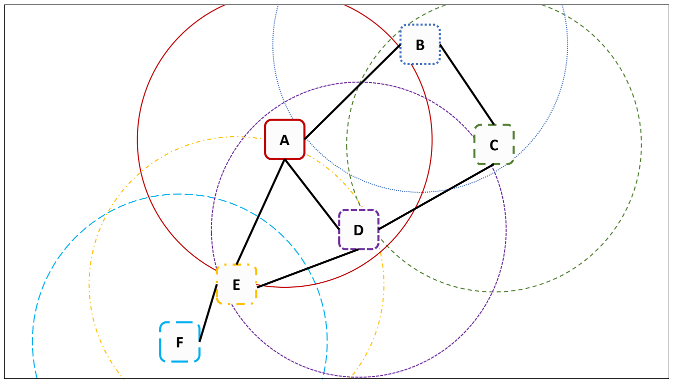

Fig. 2. Network Topology

In order to evaluate our SDN framework, the network would have to feature several multi-hop paths between pairs of nodes so that the controller's routing decisions could have an impact. In order to achieve this multi-hop architecture in our lab's restricted area, we attached multiple RF signal attenuators to the wireless interfaces and configured them to use a single antenna, therefore reducing signal quality without physically distancing the nodes. The nodes were eventually arranged in the static topology represented in Figure 2. The wireless cards of the nodes were configured in Ad-Hoc mode, meaning that the nodes had connectivity with the one-hop neighbors, but there was no routing algorithm by default to allow multi-hop connections.

TABLE II
OBSERVED METRIC VALUES AT T=0S

Link	PHY	PDR	Ut	w	DAT
A-B	MCS 5	0.9	0.27	0.026	52.02 Mbits/s
A-D	MCS 7	1	0.27	0.019	72.2 Mbits/s
B-C	MCS 7	0.78	0.12	0.020	56.32 Mbits/s
D-C	MCS 5	0.91	0.25	0.025	52.60 Mbits/s

TABLE III
OBSERVED METRIC VALUES AT T=15S

Link	PHY	PDR	Ut	w	DAT
A-B	MCS 5	0.82	0.38	0.034	49.13 Mbits/s
A-D	MCS 7	0.98	0.38	0.023	70.03 Mbits/s
B-C	MCS 7	0.72	0.14	0.022	50.54 Mbits/s
D-C	MCS 5	0.8	0.4	0.036	46.24 Mbits/s

B. *Contending Link Experiment*

In this proof-of-concept scenario, we aim at investigating the routing decision performed by the MC for establishing a multi-hop path for a flow between a pair of nodes, in the case where a contending flow has been activated between nodes that are not directly involved in the path. All nodes are configured to operate in a 20 MHz channel. We compare the decision of our SDN framework with the one of the widely adopted OLSRv2. OLSRv2 uses the Directional Airtime Metric (DAT) [22] in order to characterize the quality of the established links. In short, this metric is a successor of the ETX metric and essentially takes into account the link speed, via an external measurement process and the packet loss rate through analysis of multicast control traffic. DAT can be expressed with a human readable value of link speed in bits/s, between 119 bit/s and 2 Gbit/s.

We conducted two experiments, one with the SDN framework and one with the OLSR, both under the same scenario. We start the scenario with the activation of a UDP flow between nodes A and C. The data rate of the flow is 30 Mbit/s and is enough to saturate both the two-hop links between A and C. The first refers to the path through relay node B and the second through relay node D. In Table II we show the observed metrics as collected by the respective modules at *t=0s*. Although the network is essentially idle, the airtime utilization values are considerably over zero, as they represent the time for the transmission of SDN/OLSR control messages as well as 802.11 control and management frames. However, the link between B and C exhibits lower utilization due to the nodes being more isolated. By inspecting our defined weights (w) and the DAT values is evident that both SDN and OLSR will prefer to route the flow through node D and this is exactly what happened. Figure 3 shows the achieved throughput of the flow for the period that it was active for both SDN and OLSR. In *t=10s* we stop the flow and we activate a new flow of 10 Mbit/s UDP traffic between node E and F, which are directly connected. After 5s, in *t=15s*, which is enough time for the metrics' values to be updated, we activate again the flow between A and C. The observed metric values are referenced in Table III.

The inspection of the table allows us to derive some very useful insights. We observe that the utilization values are significantly increased for both nodes A and D, while for node B it remains quite similar with the case where the flow between A and C was the only one existing in the network. The transmission PHY rate as well as the PDR remain at similar values for all the links involved. This can be attributed to the fact that node E is in the sensing range for both nodes A and D, resulting in both nodes spending significant time sensing the medium busy. However, as both nodes are also in the range of node E, when it is their turn to transmit, E will sense the medium busy and defer its transmission. This explains why PDR remains unaffected, as the carrier sense of 802.11 CSMA/CA guarantees a minimum, close to zero, amount of collisions and consequently, the rate control algorithm keeps the same selection of PHY rate. This is the same reason why OLSR that considers only link speed and packet losses, calculates similar values for the DAT metric, as it does not account for the utilization of the links. Observing the weights we can deduce that the SDN controller selects the path through node B, while OLSR does not change its selection of a path through node D. Eventually, Figure 3 shows the achieved throughput for the last and the previous stages of the scenario for both SDN and OLSR. A major throughput improvement, around 41%, in this scenario, is evident when routing accounts for the utilization of the wireless medium, as in the case of our SDN framework.

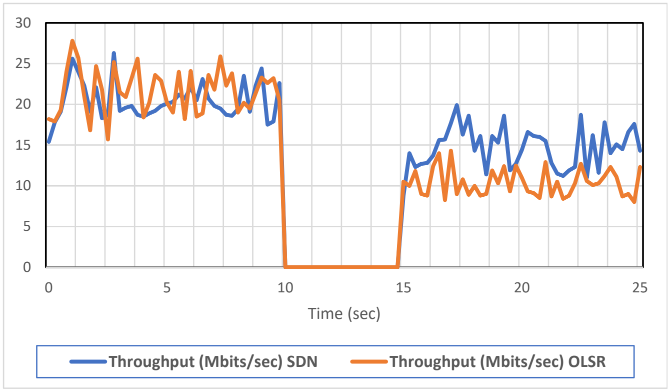

Fig. 3. Throughput of flow between A and C.

V. Conclusion

In this paper, we detailed the implementation of our SDN framework for an 802.11 MANET network that effectively adopts and combines the strengths of traditional MANET routing protocols and SDN architectures. Our developed solution is fault-tolerant and resilient with each node ready to assume the role of the SDN controller for the whole network, through a Raft-based election process. It features a distributed topology discovery process and is based on an in-band architecture allowing the applicability on devices with a single wireless interface. Furthermore, it follows a cross-layer approach in routing by leveraging 802.11 MAC layer statistics in order to accurately capture link capacities. The benefits of such an approach have been demonstrated under a proof-of-concept experimentation.

As future work, we consider the introduction of hierarchy and multiple tiers of SDN networks that will enable our system to scale efficiently.

References

[1] MarketsandMarkets, "Software-Defined Networking Market by Component (SDN Infrastructure, Software, and Services), SDN Type (Open SDN, SDN via Overlay, and SDN via API), End User, Organization Size, Enterprise Vertical, and Region - Global Forecast to 2025," 2020.

[2] T. Clausen and P. Jacquet, "Optimized link state routing protocol (OLSR)," Tech. Rep., 2003.

[3] A. Detti, C. Pisa, S. Salsano, and N. Blefari-Melazzi, "Wireless mesh software defined networks (wmSDN)," in *2013 IEEE 9th international conference on wireless and mobile computing, networking and communications (WiMob)*, 2013.

[4] M. Labraoui, M. Boc, and A. Fladenmuller, "Self-configuration mechanisms for SDN deployment in Wireless Mesh Networks," in *2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2017.

[5] "Flexible SDN control in tactical ad hoc networks, author=Poularakis, Konstantinos and Qin, Qiaofeng and Nahum, Erich M and Rio, Miguel and Tassiulas, Leandros," *Ad Hoc Networks*, vol. 85, pp. 71–80, 2019.

[6] D. Giatsios *et al.*, "Giatsios, Dimitris and Choumas, Kostas and Flegkas, Paris and Korakis, Thanasis and Cruelles, Joan Josep Aleixendri and Mur, Daniel Camps," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, 2019.

[7] C. Y. Hans, G. Quer, and R. R. Rao, "Wireless SDN mobile ad hoc network: From theory to practice," in *2017 IEEE International Conference on Communications (ICC)*, 2017.

[8] P. Bellavista, A. Dolci, and C. Giannelli, "MANET-Oriented SDN: motivations, challenges, and a solution prototype," in *2018 IEEE 19th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, 2018.

[9] K. Streit, N. Rodday, F. Steuber, C. Schmitt, and G. D. Rodosek, "Wireless SDN for highly utilized MANETs," in *2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 2019.

[10] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The Design and Implementation of Open {vSwitch}," in *12th USENIX symposium on networked systems design and implementation (NSDI 15)*, 2015.

[11] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX Annual Technical Conference (Usenix ATC 14)*, 2014.

[12] M. Rademacher, F. Siebertz, M. Schlebusch, and K. Jonas, "Experiments with OpenFlow and IEEE802. 11 Point-to-Point Links in a WMN," *ICWMC 2016*, p. 111, 2016.

[13] S. Sharma, A. Nag, P. Stynes, and M. Nekovee, "Automatic configuration of openflow in wireless mobile ad hoc networks," in *2019 International Conference on High Performance Computing & Simulation (HPCS)*, 2019.

[14] O. S. Specification, "OpenFlow Version 1.3.1," *Open Networking Foundation*, 2012.

[15] X. Chen, T. Wu, G. Sun, and H. Yu, "Software-defined MANET swarm for mobile monitoring in hydropower plants," *IEEE Access*, vol. 7, pp. 152 243–152 257, 2019.

[16] I. Syrigos, S. Keranidis, T. Korakis, and C. Dovrolis, "Enabling wireless lan troubleshooting," in *International Conference on Passive and Active Network Measurement*, 2015.

[17] I. Syrigos, N. Sakellariou, S. Keranidis, and T. Korakis, "On the employment of machine learning techniques for troubleshooting WiFi networks," in *2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, 2019.

[18] D. N. da Hora, K. Van Doorselaer, K. Van Oost, R. Teixeira, and C. Diot, "Passive wi-fi link capacity estimation on commodity access points," in *Traffic Monitoring and Analysis Workshop (TMA) 2016*, 2016.

[19] Ryu Controller. [Online]. Available: https://ryu.readthedocs.io

[20] Etcd. [Online]. Available: https://etcd.io/

[21] K. Choumas and T. Korakis, "On using Raft over Networks: Improving Leader Election," *IEEE Transactions on Network and Service Management*, 2022.

[22] H. Rogge and E. Baccelli, "Directional airtime metric based on packet sequence numbers for Optimized Link State Routing Version 2 (OLSRv2)," Tech. Rep., 2016.