Lasagna: Programming Abstractions for End–to–End Slicing in Software–Defined WLANs

Estefanía Coronado*‡, Roberto Riggio‡, José Villalón* and Antonio Garrido*
*High-Performance Networks and Architectures (RAAP). University of Castilla–La Mancha, Albacete, Spain
Email: {Estefania.Coronado, JoseMiguel.Villalon, Antonio.Garrido}@uclm.es
‡FBK CREATE-NET, Trento, Italy
Email: rriggio@fbk.eu

***Abstract*—Current 802.11–based WLANs are asked to support an ever increasing number of services and applications, each of them characterized by a diverse set of requirements in terms of bitrate, latency, and reliability. Network virtualization and programmability are two emerging trends that can support the realization of such a vision in a cost–effective fashion. In this paper we introduce *Lasagna*, a novel end–to–end solution that enables flexible management of slices encompassing both the wired and the wireless segments of an Enterprise WLAN. *Lasagna* allows flexible management of network slices to meet their respective service requirements. An experimental evaluation carried out over a real–world testbed shows that *Lasagna* can ensure both functional and performance isolation between the different slices and efficient radio resource utilization. We release the entire implementation including the controller and the datapath under a permissive license for academic use.**

***Index Terms*—Network Virtualization, Slicing, Software Defined Networking, WLANs, IEEE 802.11.**

I. Introduction

Users and applications have different performance requirements in terms of bandwidth, latency, and data rate. This calls for a service–oriented approach for network resource provisioning. For example, ITU identifies three classes of services, namely enhanced mobile broadband, massive machine type communications, and ultra–reliable and low latency communications [1]. While the ITU white paper addresses specifically 5G networks, it is widely acknowledged that such networks will heavily rely on Wi–Fi as traffic offloading solution. As a matter of fact, it is expected that Wi–Fi and mobile traffic will account for 63% of the entire IP traffic by 2021 [2].

Even just for the Wi–Fi segment, a single rigid network architecture will not be enough to support the diverse and dynamic set of services and applications that will characterize future 5G mobile systems. Conversely, to make the service–oriented 5G vision become a reality, it is mandatory to abstract the physical network into multiple *end–to–end* virtual logical networks or slices, one for each service category. Software–Defined Networking (SDN) and Network Function Virtualization (NFV) are considered to be two of the most promising technological enablers for achieving this vision. However, while several proposals have been put forward for the mobile core [3], [4], [5], [6], the concept is still at its infancy in the radio access segment.

In this paper we present a programmable end–to–end network slicing framework for WLANs. This framework pursues three objectives: (i) *programmability*, we want to allow network administrators to specify how a precise portion of the flowspace shall be treated in the wireless access segment; (ii) *isolation*, we want to make sure that slices are kept isolated from the logical and the performance standpoints; and (iii) *customization*, we want to allow each slice to specify its own traffic prioritization policies, e.g. in terms of aggregation, rate selection, etc. Notice how while the IEEE 802.11e amendment [7] did introduce the concept of traffic differentiation and prioritized access, it did not provide any programming abstraction to allow end–to–end slice management. Similarly, the de–facto standard for SDN in wired networks, i.e. OpenFlow [8], does not encompass the wireless access segment.

Taking this into consideration, the contribution of this paper is threefold. First, we extend the OpenFlow match rule with some fields from the IEEE 802.11 header. Second, based on this extended match rule, we introduce a new programming abstraction named *Traffic Rule* enabling the specification of customized slicing policies for a precise portion of the flowspace. Finally, we implement a flexible hypervisor capable of ensuring the required logical and performance isolation between slices while at the same time enabling slice customization and efficient radio resource utilization. The proposed system, named *Lasagna*, has been implemented and tested on top of the 5G–EmPOWER Software–Defined Radio Access Network platform [9]. An experimental evaluation encompassing a wide range of usage scenarios and conducted over a real–world testbed has demonstrated the capability of *Lasagna* to meet the design requirements. We release the entire implementation under a permissive APACHE 2.0 license for academic use[1].

The rest of this paper is structured as follows. In Sec. II we cover the related work. We delve into the *Lasagna* design aspects in Sec. III, whereas in Sec. IV the implementation details are presented. Section V and Sec. VI discuss, respectively, the evaluation methodology and reports on the outcomes of the measurements campaign. Finally, Sec. VII draws the conclusions pointing out future work.

*This work was done while Estefanía Coronado was a member of the University of Castilla–La Mancha, Albacete, Spain.

978-1-5386-4725-7/18/$31.00

[1]Online resources available at: http://empower.create-net.org/

II. Related Work

The IEEE 802.11e amendment [7] established the foundations for traffic prioritization through the Enhanced Distributed Channel Access (EDCA) function. Nevertheless, it does not provide any API for end–to–end service management nor ensures radio resource isolation between traffic classes [10].

In this sense, queuing management has claimed significant research efforts. H. Luo et al. [11] introduce an optimized scheduling scheme that allocates the transmission opportunities for each station through a quadratic performance index based on the lenght of the traffic queues. However, although the packet delay is minimized, a fixed transmission rate and a single flow per station are assumed. The same strategy is studied using fuzzy logic in [12]. The inter–node priority is assigned to each station based on the Access Category (AC), the collision rate, and the residual energy level.

Significant QoS improvements can be achieved in 802.11n [13]. This amendment makes it possible to reduce channel contention and protocol header overhead by introducing frame aggregation, thus improving the radio resource utilization [14], [15]. S. Seytnazarov et al. [16] optimize voice traffic performance using Aggregated MAC Protocol Data Unit (A–MPDU) aggregation based on QoS requirements and end–to–end delay statistics. The improvement is however achieved at the expense of the remaining traffic classes. Multimedia applications are the target of E. Charfi et al. [17]. This scheme uses three priority queues, and frames are transmitted according to the lowest time to serve a packet (i.e. real–time frames have lowest serving times) and the amount of time waiting in the scheduler. The channel utilization can be further optimized if only the first Ethernet header is kept in an Aggregate MAC Service Data Unit (A–MSDU) since the remaining ones are equal [18]. Nevertheless, this modification in the frame structure makes the approach non–standard compliant.

Given that ensuring QoS provisioning may penalize low priority traffic, some works pursue fairness as additional performance metric. On this basis, A–MPDU aggregation from different traffic classes in the same frame is proposed in [19]. This mechanism inserts as many high priority packets as possible in the aggregated frames, as well as lower priority frames. However, multi–priority aggregation is not supported by the 802.11 standard. O. Panova et al. [20] introduce an algorithm that switches the number of traffic classes depending on the network status.

The aggregation problem has also been widely studied from the analytical point of view. D. Kim et al. [21] present a numerical model that determines both the number of frames to be aggregated and the transmission rate according to the signal strength of the ACKs. In [22] a Discrete–Time Markov Chain is presented with the aim of improving the channel utilization and increasing the number of supported clients without increasing the aggregation delay.

Although a significant part of the studies focuses on throughput improvements, ensuring delay bounds is a more challenging task, especially for real–time critical transmissions. The proposal in [23] describes a fair scheduling algorithm for A–MSDU aggregation based on both the lifetime and the priority of the frames. By inserting high and low priority traffic in the same frame, it aims to ensure transmission fairness while reducing the average delay and the packet loss ratio. This objective is also pursued in [24] for alleviating the fairness problem. The scheme proposed by S. Kim et al. dynamically sets the duration of the transmission opportunities for each traffic class based on the delay constraints of the frames and the network load. Nevertheless, the work is mainly targeted at multimedia traffic. Furthermore, the current network load must be reported to the stations through beacon frames, hence this proposal being non-standard compliant.

Despite the improvements achieved, the previous approaches do not allow to define customized traffic management policies across different users, applications and services [25], [26], [27], [28]. In this regard, J. Saldana et al. simulate the expected behaviour of a central controlled WLAN with EDCA support [29]. In this work, frame aggregation is disabled if VoIP traffic is found. However, although it evaluates the impact of two values for the size of the A–MPDU, it cannot be dynamically modified. Furthermore, the analysis just emulates the expected behaviour of the network. K. Nakauchi et al. propose an airtime–based resource allocation mechanism for network virtualization [30]. A similar idea is implemented on a real testbed in [31]. Here, the scheduler manages both the network slices and the clients in each slice depending on the traffic requirements and the loss ratio. However, slice isolation is not ensured given that the performance drop in a client may degrade the throughput of the clients in other slices. Finally, drawing an analogy with the 5G networks, the work in [32] presents a slicing solution for Wi–Fi Access Points (APs) based on the airtime consumed as the resource to share. Nevertheless, the proposal is only assessed via simulation.

It should be noted that most of the works in the literature, if not all, propose point solutions to very specific use cases and problems. Conversely, our standpoint is that a one–size–fits–all architecture is unlikely to meet the needs of current and, especially, future use cases. As a consequence, realizing the needed service–oriented vision requires a flexible and programmable network architecture spanning both the wired and the wireless network segments. In this paper we precisely address this challenge for 802.11–based enterprise WLANs. To the best of our knowledge there is no other work proposing a programmable and dynamic end–to–end network slicing framework that takes into account resource isolation, and that is implemented and tested over a real–world WLAN.

III. Lasagna Overview

Supporting the requirements of current services and applications can often result in far–reaching changes in both the network architecture and the protocol stacks. The emerging SDN paradigm aims to break free of this constraint by decoupling the data–plane and the control–plane. The network intelligence is then shifted from the network devices to a central location (the network controller) allowing to implement sophisticated

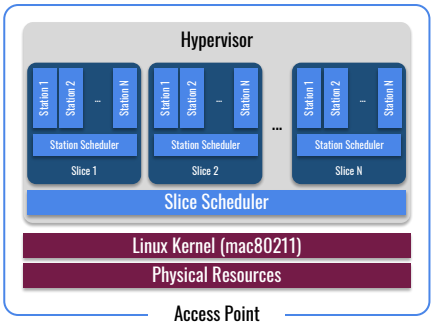


Fig. 1. The *Lasagna* hypervisor architecture.

traffic management policies based on the global network view exposed by the controller, while the devices just apply the rules defined at the control plane.

This paper extends the mainstream SDN network slicing concept to the wireless access segment in the specific case of 802.11–based WLANs. The proposed framework, *Lasagna*, aims at ensuring efficient sharing of the same physical infrastructure by different services and applications. More specifically, we assume that an infrastructure provider owns the physical Wi–Fi APs and the network switches, which are in time leased to the service provider (the slice's owners). Notice how details about pricing although important are out of the scope of this paper. Moreover, although this work focuses on Wi–Fi networks, *Lasagna*'s design principles are quite general and can be easily extended to other radio access technologies.

The basic *Lasagna* design builds upon a programmable hypervisor sitting on top of the standard Linux Wi–Fi stack. The hypervisor is in charge of creating, monitoring, and managing the network slices, thus ensuring performance isolation and efficient radio resource utilization. The high–level architecture of the hypervisor is depicted in Fig. 1. As can be seen, each AP can support a variable number of slices. Each of these slices contains one aggregation buffer for each Wi–Fi client in the network and can have its own EDCA parameters. For example, one slice can use no aggregation and voice–optimized EDCA parameters, while another slice can enable frame aggregation and use background traffic EDCA parameters.

A. The Traffic Rule Abstraction

A network slice is defined as a set of radio resources that are assigned to a particular flow. In this way, a network slice can be simultaneously shared among multiple services and applications, and a client can make use at the same time of different network slices. The framework introduced in this work provides the ability to create programmable network slices. To this end, a new abstraction named *Traffic Rule* is introduced to map a specific portion of the flowspace to a particular scheduling discipline. The *Traffic Rule* abstraction defines a set of parameters the AP must use when forwarding traffic belonging to a particular slice. Such parameters include:

- *EDCA*. The EDCA parameters to be used for this slice. This includes Congestion Window (CW), Arbitration Interframe Space Number (AIFSN) and Transmit Opportunities (TXOPs).
- *Aggregation*. The type of frame aggregation to be used for this slice, including A–MSDU, A–MPDU, or none.
- *Quantum*. The fraction of airtime that can be assigned to this slice in each round.

Any parameter of a *Traffic Rule* can be modified in runtime by the controller. Furthermore, the *Traffic Rule* abstraction can be combined with the *Transmission Policy* abstraction [33], which allows the SDN controller to specify the range of parameters the AP can use for its communication with a wireless client. Such parameters include:

- *MCSes*. The set of Modulation and Coding Schemes (MCSes) that can be used by the rate selection algorithm.
- *RTS/CTS Threshold*. The frame length above which the RTS/CTS handshake must be used.
- *No ACK*. The AP shall not wait for ACKs if true.
- *Multicast policy*. Specifies the retransmission policy used in case of multicast traffic, which can be Legacy, Direct Multicast Service (DMS), or Unsolicited Restries (UR).
- *UR Count*. Specifies the number of UR retransmissions.

Transmission Policy configurations can be specified on a L2 destination address basis. As a result, for each destination address and for each slice in the network, a specific *Transmission Policy* configuration can be created. The *Transmission Policy* abstraction allows the controller to specify the set of MCSes that can be used by the rate control algorithm. However, notice that the frame–by–frame MCS selection is implemented at the AP and not at the controller.

A *Traffic Rule* configuration is identified by the tuple $(SSID, DSCP)$, where the Service Set Identifer (SSID) refers to the name of a Wi–Fi network and the Differentiated Services Code Point (DSCP) determines the priority of each IP packet. The next subsection will explain how a precise portion of the flowspace can be assigned to a *Traffic Rule*. As can be seen, using both the *Traffic Rule* and the *Transmission Policy* abstractions it is possible to ensure that applications and services with the same requirements are provided the same network resources. Notice how a default *Traffic Rule* with DSCP set to $0x00$ is always present.

B. End–to–end Slicing

Forwarding policies in OpenFlow–enabled switches can be configured by a logically network controller by specifying a set of rules. Each rule is composed of a *Match*, used to identify the flow, and an *Action*, which specifies the operation to be performed on each packet in that flow, e.g. forward to zero, one, or more output ports, add/remove a header, set a field, etc. The fields that can be used for the *Match* rule compose the so–called OpenFlow *Extended header* and include a combination of link, network, and transport header fields. Figure 2 shows some examples of traffic matches that can be defined using the OpenFlow *Extended header*. The first match includes the traffic with the IP DSCP field set to zero. The second rule includes HTTP traffic with $0x40$ as IP DSCP.

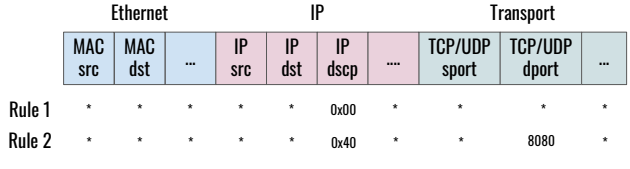


Fig. 2. The OpenFlow *Extended header*.

Traffic matching a given OpenFlow rule is tagged with a unique DSCP value and then dispatched to the Hypervisor. Here, the tuple $(SSID, DSCP)$ is then used to redirect the traffic to the correct slice. Notice how, since a Wi–Fi station can be associated to one, and only one, SSID at any given time, the SSID can be easily computed using a packet MAC destination address.

C. Airtime–based Slice Scheduling

The hypervisor proposed in this work uses a modified version of the Deficit Round Robin policy to schedule *Traffic Rules*. The proposed scheduling policy, named Airtime Deficit Weighted Round Robin (ADWRR), assigns to each *Traffic Rule* a fraction of the airtime according to its relative priority. This is done because, in a wireless network, the *cost* of transmitting a frame depends on the frame length *and* on the actual channel conditions experienced by the receiver of that frame. For example, a receiver that is far away from the AP will utilize more radio resources due to the use of less efficient MCSes and/or more frame retransmissions.

Our hypervisor uses the transmission statistics maintained by the AP MCS selection module in order to estimate the airtime that will be required to serve a particular *Traffic Rule*. The details of the MCS selection algorithm used by the AP are not important as long as they include, for each client, the link delivery probability for each MCS supported by the AP. In our implementation we relied on the Minstrel algorithm [34], which is available in the Linux kernel. Based on this algorithm, let $P(R_i)$ be the probability of transmitting a frame and receiving the corresponding Wi–Fi acknowledgement using the MCS R_i, and let R_{best} be the MCS with the highest delivery probability. The expected transmission airtime A for a packet L bits long can be approximated with:

$$A = \frac{1}{P(R_{best})}\Big(DIFS + \frac{L}{R_{best}} + SIFS + T_{ack}\Big)$$

Notice how this formula ignores the exponential growth of the congestion window for each failed transmission. Such a simplification proved to be sufficient in our measurement campaign. Nevertheless, a better estimation of the transmission airtime can be computed using the results presented in [35].

The pseudo code of the enqueue and dequeue processes used by the hypervisor are given respectively in Alg. 1 and Alg. 2. Variables and data structure exploited by both algorithms are summarized in Table I.

The hypervisor maintains a list of currently backlogged *Traffic Rules* ($ActiveQueue$). Incoming frames are classified according to the $(SSID, DSCP)$ tuple (Alg. 1, row 3) and then fed to the corresponding *Traffic Rule* (Alg. 1, row 5). If such a *Traffic Rule* does not exist yet, it is created dynamically by the hypervisor.

At each round the deficit counter $DC(i)$ of the current *Traffic Rule* is increased by a fixed quantity $Q(i)$ (Alg. 2, row 5). The hypervisor only serves *Traffic Rules* whose expected transmission time is smaller than the deficit counter (Alg. 2, row 8). After each transmission, this counter is decreased by the expected transmission time of the frame (Alg. 2, row 11). A frame whose transmission time exceeds the deficit counter is held back until the next visit of the scheduler (Alg. 2, row 13). Empty *Traffic Rules* are removed from the $ActiveQueue$ and their deficit counter is set to zero (Alg. 2, row 15).

It should be noted that the possibility of assigning a different quantum to each *Traffic Rule*, and thus to each slice, enables advanced QoS management features. For example, the slices supporting services with stricter performance requirements can be assigned more radio resources by specifying a larger value for the *Traffic Rule* quantum parameter.

Each *Traffic Rule* contains multiple *Aggregation Buffers*, one for each station in the virtual network. *Aggregation Buffers* share the DC of the parent *Traffic Rule*. Nevertheless, each

TABLE I
HYPERVISOR DATA STRUCTURES

Variable	Default	Description
$ActiveQueue$	$\{\emptyset\}$	List of backlogged *Traffic Rule*.
$Q(i)$	$12000\mu s$	*Traffic Rule* i Quantum.
$DC(i)$	0	*Traffic Rule* i Deficit counter.

Algorithm 1 Enqueuing process.

```
1: procedure ENQUEUE(p)
2:     (SSID, DSCP) ← GetTrafficRule(p)
3:     if (SSID, DSCP) not in ActiveQueue then
4:         ActiveQueue.pushBack((SSID, DSCP))
5:     ActiveQueue((SSID, DSCP)).enqueue(p)
```

Algorithm 2 Dequeuing process.

```
1: procedure DEQUEUE
2:     (SSID, DSCP) ← GetTrafficRule(p)
3:     if ActiveQueue is not empty then
4:         i = ActiveQueue.next()
5:         DC(i) = DC(i) + Q
6:         while True do
7:             airtime = ActiveQueue(i).computeTxAirtime()
8:             if airtime < DC(i) then
9:                 p = ActiveQueue(i).dequeue()
10:                p.send()
11:                DC(i) = DC(i) - airtime
12:            else
13:                break
14:        if i is empty then
15:            ActiveQueue.remove(i)
```

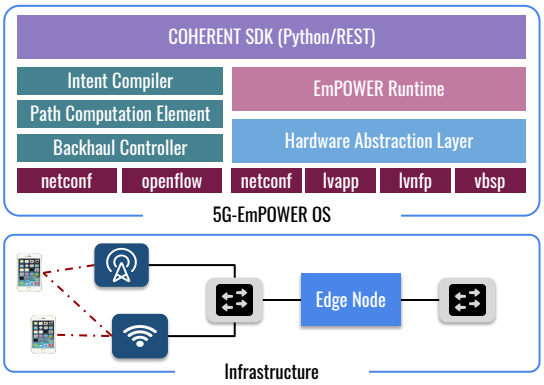


Fig. 3. The 5G–EmPOWER MEC–OS System Architecture.

Traffic Rule can schedule *Aggregation Buffers* in a different manner. For example, one *Traffic Rule* can schedule its stations using a Round Robin policy, while another *Traffic Rule* could use a best–MCS policy. Notice how, according to the configuration of the parent *Traffic Rule*, *Aggregation Buffers* can generate A–MSDU, A–MPDU, or non–aggregated frames. In case of aggregated frames, the maximum frame length and the aggregation timeout can be on a per–*Traffic Rule* basis.

IV. Implementation Details

A. Overview

The implementation of *Lasagna* has been carried out taking as a reference the 5G–EmPOWER plaform [9]. 5G–EmPOWER is a Multi–access Edge Computing Operating System (MEC–OS) supporting lightweight virtualization and heterogeneous radio access technologies. The high–level architecture of this platform is shown in Fig. 3.

5G–EmPOWER builds upon a hardware abstraction layer covering several radio access technologies, such as Wi–Fi and LTE. Furthermore, it defines a set of high–level programming abstractions that are exposed to the application layer using a Python–based Software Development Kit (SDK).

B. Control and Data Plane Implementation

Each AP in a 5G–EmPOWER–managed network consists of two components: one OpenvSwitch [36] instance managing the communication over the wired backhaul, and one Click modular router [37] instance implementing the 802.11 data–path. Click is a framework for writing multi–purpose packet processing engines and is used to implement just the wireless client/AP frame exchange.

The network intelligence is implemented at the 5G–EmPOWER controller, which is in communication with the APs in the data–plane through its southbound interface using a persistent TCP connection. The protocol used for this communication is outside the scope of this paper. A full account of its features can be found online [38].

Similarly, the OpenvSwitch running within each AP operates under the supervision of an OpenFlow–enabled backhaul controller (Ryu in this particular case [39]). The intent–based

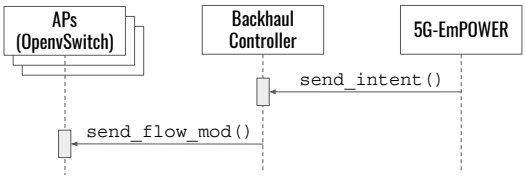


Fig. 4. *Traffic Rule* creation process.

interface presented in [40] is used for the communication between 5G–EmPOWER and the backhaul controller. Such interface has been extended in order to allow the 5G–EmPOWER controller to request the backhaul controller to tag with a particular DSCP code all the traffic matching a certain flow rule and arriving on the backhaul interface of a given AP.

C. Traffic Rule Creation

The *Traffic Rule* abstraction is exposed to the application layer through an object mapping properties to operations. This allows it to manipulate the *Traffic Rule* configurations defined for a certain slice by simply accessing the `traffic_rules` property of a *Tenant* object (i.e. a virtual network). For example, defining a new *Traffic Rule* configuration for HTTP traffic can be as simple as shown below.

```
>>> tenant.traffic_rules["tp_dst=8080"] = \
>>>     TrafficRule(tenant, dscp=0x40, quantum=5000)
```

The listing above will trigger two operations: (i) the creation of a new slice for the DSCP $0x40$ at every AP in the virtual network using 5000 μs for the slice quantum; and (ii) a message to the backhaul controller through the intent–based networking interface, as depicted in Fig. 4. This message has the following structure:

```
{
  "src_dpid": "00:00:00:00:00:0A",
  "src_port": 1,
  "dst_dpid": "00:00:00:00:00:0A",
  "dst_port": 4,
  "matches": {
    "tp_dst": "8080"
  },
  "tag": 0x40
}
```

The pair (src_dpid, src_port) identifies the AP backhaul port, while the pair (dst_dpid, dst_port) identifies the virtual port to which the Click instance implementing the Wi–Fi data–path is attached. The semantic of the message is that all the Wi–Fi–bound traffic arriving on the AP backhaul port that matches the specified rule must be tagged with the specified DSCP code. This will allow the hypervisor to dispatch the specified portion of the flowspace to the new *Traffic Rule*.

D. Traffic Rule Monitoring and Update

The status of each *Traffic Rule* can be monitored by the 5G–EmPOWER controller using the slice telemetry framework. More specifically the controller can periodically gather the status of all the *Traffic Rules* defined in a certain AP. Such

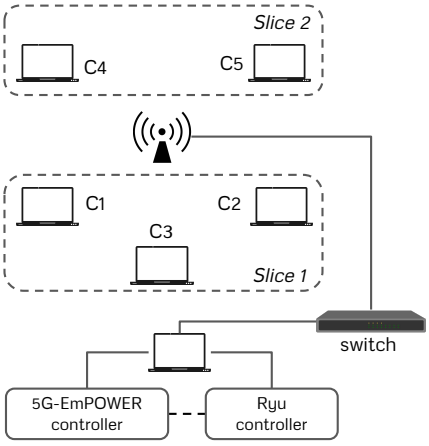


Fig. 5. Testbed deployment layout.

TABLE II
EVALUATION SCENARIOS.

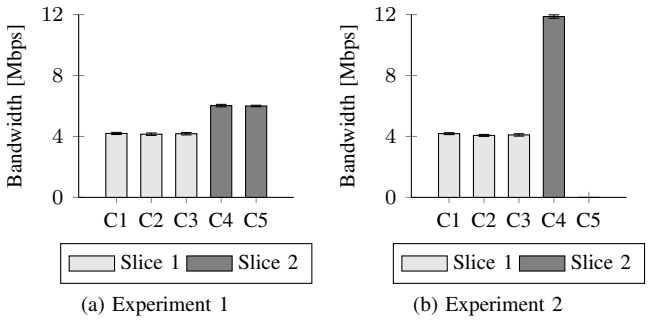
Test	Users Slice 0	Users Slice 1	DSCP Slice 0	DSCP Slice 1	Channel conditions	Traffic
B1	5	0	-	-	Equal	Const.
B2	5	0	-	-	Different	Const.
1, 9	3	2	0x00	0x00	Equal	Const.
2, 10	3	1	0x00	0x00	Equal	Const.
3, 11	3	2	0x00	0x20	Equal	Const.
4, 12	3	1	0x00	0x20	Equal	Const.
5, 13	3	2	0x00	0x00	Different	Const.
6, 14	3	1	0x00	0x00	Different	Const.
7, 15	3	2	0x00	0x20	Different	Const.
8, 16	3	1	0x00	0x20	Different	Const.
17	3	2	0x00	0x00	Equal	Int.
18	3	2	0x00	0x00	Different	Int.

status includes for example the current number of backlogged frames, the total airtime spent, the number of transmitted packets and bytes, and the number of dropped packets and bytes. Such information can be used for different purposes. For instance, the controller could use it to identify if some of the *Traffic Rules* are inactive, thus triggering a reassignment of the available resources to other *Traffic Rules*.

V. Evaluation Methodology

Lasagna has been evaluated on a real–world testbed whose high–level architecture is depicted in Fig. 5. The testbed is composed of one AP and five clients. A laptop connected to the wired segment of the network runs both the Ryu and the 5G–EmPOWER controllers. The AP is based on the PCEngines ALIX 2D (x86) processing board and is equipped with two Wi–Fi cards based on the Atheros AR9220 chipset. OpenWRT 15.05.01 is used as operating system for the AP. The experiments are conducted on the 5 GHz band with the cards operating on channel 48 in 802.11n mode. Two slices are created in the network: the first one consists of three users (C1, C2 and C3), while the second one consists of two users (C4 and C5). The wireless clients are located 5 m away from the AP and are standard laptops.

The evaluation comprises eighteen different scenarios divided into three groups. In all the scenarios a saturated UDP stream was generated between the laptop running the controllers and each wireless client. Each measurement was 60s long. Traffic was generated using Iperf. The results reported in the next section are the average of 10 runs. The first group of experiments (1 to 8) are carried out without any kind of AP/controller telemetry. Then, the same experiments are repeated with the slice telemetry enabled (9 to 16). In this configuration the controller periodically polls the AP to gather the slice utilization statistics. The polling period was set to 100ms. These two groups of experiments make use of constant bitrate traffic. By contrast, intermittent traffic is used in scenarios 17 and 18. Finally, the scenarios named B1 and B2 refer to the baseline configuration in which no hypervisor is used. As evaluation metrics we have used the aggregated throughput, the bandwidth achieved by each client, and the jitter. Apart from the aforementioned transmissions, no other traffic exists in the network. A summary of the tests conditions used in each scenario is provided in Table II.



(a) Experiment 1 (b) Experiment 2

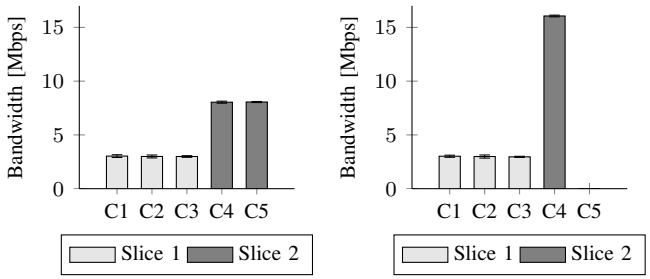
Fig. 6. Bitrate comparison for two network slices with the same priority and equal channel conditions for the clients.

VI. Results

A. Traffic Prioritization

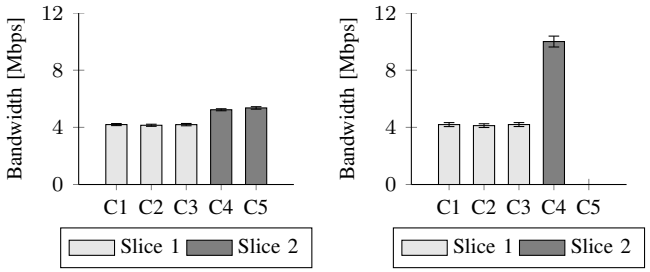
Experiments 1 and 2 aim to demonstrate the isolation capabilities across slices of our hypervisor and how the number of clients per slice does not affect the performance of other slices. In experiment 1 the slices have the same priority and the clients are in the same channel conditions (based on signal strength measurements). The *Quantum* of the slices is set to the time needed to transmit a standard Ethernet frame (1500 bytes of MAC payload) at the 6 Mbps basic rate in IEEE 802.11n. The results for these tests are reported in Fig. 6a, where it can be seen that the *Quantum* is equally divided between the clients in each slice.

Experiment 2 follows the same approach of experiment 1 with the difference that, in this case, a single client is active in the *Slice 2*. As can be observed in Fig. 6b, the slice isolation features provided by our hypervisor allow the client in the *Slice 2* to fully use the resources assigned to its slice.



(a) Experiment 3 (b) Experiment 4

Fig. 7. Bandwidth comparison for two network slices with different priority and equal channel conditions for the clients.



(a) Experiment 5 (b) Experiment 6

Fig. 8. Bandwidth comparison for two network slices with the same priority and equal channel conditions for the clients.

Moreover, it can also be noticed that the performance of the clients connected to the *Slice 1* is not affected by the resource redistribution in the *Slice 2*.

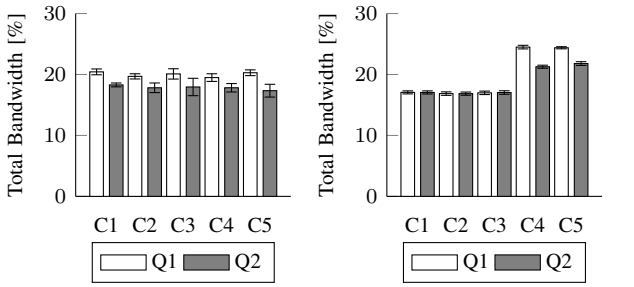
Experiments 3 and 4 present the same scenarios described in the previous measurements with the main difference that, in this case, the *Slice 2* is given twice the *Quantum* of the *Slice 1*. Figure 7 sketches this new situation, and proves how the slices isolation is guaranteed along with the priority assignment.

B. Performance Isolation

These experiments aim to show the hypervisor performance isolation features when some clients experience poor channel conditions. We remind the reader that the hypervisor follows an ADWRR scheduling discipline. Therefore, the transmission opportunities of the users depend on the required transmission time, and hence, on their channel conditions.

The first set of experiments (tests 5 to 8) demonstrates how the hypervisor can ensure the coexistence of the slices even when some users have poor channel conditions. Figure 8 plots the results in terms of bandwidth for experiments 5 and 6. The test conditions are the same than in experiments 1 and 2, with the difference that the station C4 is placed 25 m away from the AP. In Fig. 8 it is shown that the *Slice 1* does not present changes with respect to the first scenarios and is not affected by the problems found in the other slice. Moreover, in Fig. 8b it can be observed that when only the station C4 is connected to the *Slice 2*, its performance is, due to the poor channel status, lower than the one shown in Fig. 6b. As a result, the other users in the same slice are also affected by this issue, as is the case of the station C5. However, the throughput of the *Slice 1* is still unaffected. Tests 7 and 8 present the same results but having increased the priority of the *Slice 2*. For this reason, and due to space constraints, the graphical results of these tests are omitted.

In a system without slicing capabilities all the stations will equality share the available radio resources only if they experience similar channel conditions. If such conditions are not met, the performance of all the users in the network will be penalized due to the so called *IEEE 802.11 Performance Anomaly* [10]. Figure 9a illustrates this situation by using two different channel qualities. In the scenario named Q1 all



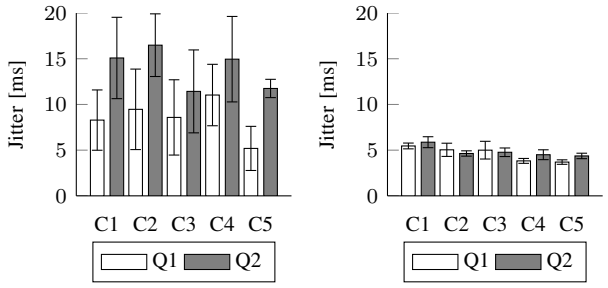
(a) Baseline (Tests B1–B2) (b) Proposal (Tests 1–5)

Fig. 9. Bitrate comparison between a baseline scenario and *Lasagna* using two slices with the same priority.

the users experience similar channel conditions, while in the scenario marked as Q2 the station C4 experiences poor channel conditions. This issue can be also verified in Fig. 10a, where it is shown how the transmission jitter for all the stations increases when a user experiencing poor channel conditions is added to the network. By contrast, as can be observed in Fig. 9b, by using our hypervisor only the slice with the client having worse channel conditions is penalized. Furthermore, the transmission jitter is also maintained for the stations in the *Slice 1*, while the one in the *Slice 2* is just slightly increased when these issues appear (Fig. 10b).

C. Resource Reallocation

Experiments 17 and 18 aim at demonstrating the ability of the system to dynamically reallocate radio resources when facing traffic changes. To this end, in experiment 17 the stream addressed to the client C5 stops after 30s. In Fig. 11 it can be seen that when the transmission finishes, the resources are dynamically reassigned to the remaining clients in the same slice. Similar results are shown in Fig. 12 for experiment 18, which differs from the previous one in the fact that the client C4 experiences poor channel conditions. As can be observed, when the stream addressed at the client C5 stops, the resources of the *Slice 2* are entirely assigned to the client C4. This is because the client C4 is the only station connected to the *Slice 2*. Conversely, in the case of a higher number of clients, the resources would be equally allocated among them.



(a) Baseline (Tests B1–B2) (b) Proposal (Tests 1–5)

Fig. 10. Jitter comparison between a baseline scenario and *Lasagna* using two *Traffic Rules* with the same priority.

TABLE III
CPU AND MEMORY USAGE RESULTS.

	CPU [%]		**Memory [%]**	
	μ	σ	μ	σ
Baseline	77.579	4.946	23.184	0.211
Proposal without telemetry	79.572	7.658	26.637	0.735
Proposal with telemetry	80.481	6.579	30.559	0.260

D. CPU and Memory Consumption

In this set of experiments we aim to show that the computational overhead introduced by our hypervisor does not penalize the data–path performance nor incurs in high CPU and/or memory utilization. In this regard, Table III compares the CPU and memory utilization of the baseline system with the ones of the system using the proposed hypervisor with and without the slice telemetry. These data are collected once per second. As can be seen, the difference found in the CPU consumption is practically negligible. Also, the increase in the memory consumption without introducing slice telemetry is just around 3%, which increases to 7% when the telemetry is used. Finally, the impact of the slice telemetry on the data–path performance is always lower than 1%.

E. Slice Telemetry

As stated in the previous section, experiments from 1 to 8 have been performed with the slice telemetry disabled, while experiments from 9 to 16 were performed with the slice telemetry enabled. However, the results difference is practically negligible and always lower than 1% (which is a value within the confidence interval of the experiments). Therefore, due to space constraints, only the results without telemetry are shown for all the experiments.

VII. CONCLUSIONS

In this paper we propose, *Lasagna*, a programmable end–to–end slicing framework for 802.11–based WLANs. We introduce the *Traffic Rule* abstraction mapping a precise portion of the flowspace to a certain scheduling discipline. An experimental evaluation performed on a real–world testbed and involving a wide range of scenarios has shown the resource allocation and performance isolation features of *Lasagna*.

As future work, we plan to implement new *Aggregation Buffers* scheduling disciplines and, at the same time, to allow the controller to swap such disciplines in runtime. Moreover, we also plan to use the framework to jointly optimize user association and radio resource utilization across the entire network. Finally, we intend to extend the proposed abstraction and its implementation to LTE networks.

ACKNOWLEDGEMENTS

This work has been supported by the Spanish Ministry of Economy under Grant Agreement BES-2013-065457, by the European Union's MINECO/FEDER funds under project TIN2015-66972-C5-2-R, by the H2020 Research and Innovation Action under Grant Agreement H2020-ICT-671639 (COHERENT), and by the EIT Digital project (16242-18) ICARO-EU "Seamless Direct Air-to-Ground Communication in Europe".

REFERENCES

[1] ITU-R, "IMT Vision - Framework and Overall Objectives of the Future Development of IMT for 2020 and Beyond," Sep. 2015.

[2] Cisco White Paper, "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016-2021," 2017.

[3] R. Santos and A. Kassler, "A SDN controller architecture for Small Cell Wireless Backhaul using a LTE Control Channel," in *Proc. of IEEE WoWMoM*, Coimbra, Portugal, 2016.

[4] A. Basta, A. Blenk, K. Hoffmann, H. J. Morper, M. Hoffmann and W. Kellerer, "Towards a Cost Optimal Design for a 5G Mobile Core Network Based on SDN and NFV," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 1061–1075, 2017.

[5] A. Banerjee, R. Mahindra, K. Sundaresan S. Kasera and J. Van der Merwe and Sampath Rangarajan, "Scaling the LTE Control-Plane for Future Mobile Access," in *Proc. of IEEE CoNEXT*, Heidelberg, Germany, 2015.

[6] X. Jin and E. Li, L. Vanbever and J. Rexford, "SoftCell: Scalable and Flexible Cellular Core Network Architecture," in *Proc. of IEEE CoNEXT*, Santa Barbara, California, USA, 2013.

[7] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 7: Medium Access Control (MAC) Quality of Service (QoS)*, ANSI/IEEE Std 802.11e, LAN/MAN Standards Committee of the IEEE Computer Society Std., 2005.

[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[9] R. Riggio, M. K. Marina, J. Schulz-Zander, S. Kuklinski and T. Rasheed, "Programming Abstractions for Software-Defined Wireless Networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 2, pp. 146–162, 2015.

[10] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda, "Performance anomaly of 802.11b," in *Proc. of IEEE INFOCOM*, San Francisco, California, USA, 2003.

[11] H. Luo and M. L. Shyu, "An Optimized Scheduling Scheme to Provide Quality of Service in 802.11e Wireless LAN," in *Proc. of IEEE ISM*, San Diego, CA, USA, 2009.

[12] V. Ilayaraja1 and R. Venkatesan, "Fuzzy Based Adaptive User-Weight Classification Scheme for EDCA in IEEE 802.11e WLAN," *International Journal of Future Generation Communication and Networking*, vol. 10, no. 1, pp. 171–186, 2017.

[13] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 5: Enhancements for Higher Throughput*, ANSI/IEEE Std 802.11n, LAN/MAN Standards Committee of the IEEE Computer Society Std., 2009.

[14] J. Kolap and S. Krishnan and N. Shaha, "Comparison of frame aggregation mechanism in 802.11n WLAN," in *Proc. of IEEE ICCICT*, Mumbai, India, 2012.

[15] Gautam D. Bhanage, "Case For Static AMSDU Aggregation in WLANs," *Computing Research Repository*, vol. abs/1707.02701, 2017.

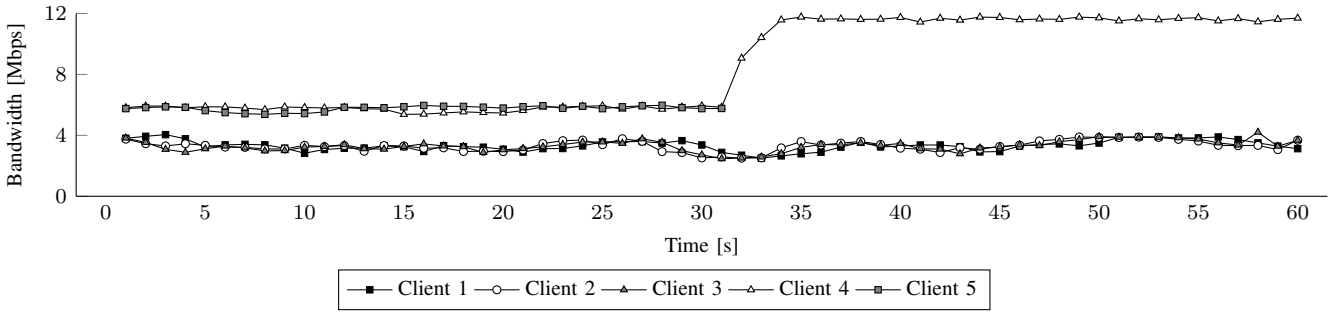


Fig. 11. Resources reassignment over time after the end of the transmission of the client C5 for equal channel conditions.

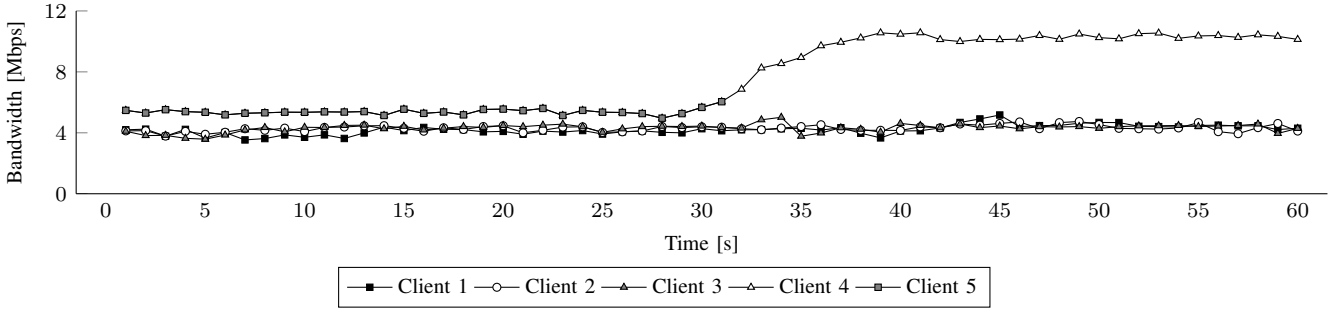


Fig. 12. Resources reassignment over time after the end of the transmission of the client C5 for different channel conditions.

[16] S. Seytnazarov and Y. T. Kim, "QoS-Aware Adaptive A-MPDU Aggregation Scheduler for Voice Traffic in Aggregation-Enabled High Throughput WLANs," *IEEE Transactions on Mobile Computing*, vol. 16, no. 10, pp. 2862–2875, 2017.

[17] E. Charfi, C. Gueguen, L. Chaari, B. Cousin and L. Kamoun, "Dynamic frame aggregation scheduler for multimedia applications in IEEE 802.11n networks," *Transactions on Emerging Telecommunications Technologies*, vol. 28, no. 2, pp. e2942–e2959, 2017.

[18] A. Saif, M. Othman, S. Subramaniam and N. Hamid, "An Enhanced A-MSDU Frame Aggregation Scheme for 802.11n Wireless Networks," *Wireless Personal Communications*, vol. 66, no. 4, pp. 683–706, Oct 2012.

[19] M. G. Sarret, J. S. Ashta, P. Mogensen, D. Catania and A. F. Cattoni, "A Multi-QoS Aggregation Mechanism for Improved Fairness in WLAN," in *Proc. of IEEE VTC*, Las Vegas, NV, USA, 2013.

[20] O. Panova and K. Obelovska, "An adaptive ACs number adjusting algorithm for IEEE 802.11 EDCA," in *Proc. of IEEE IDAACS*, Warsaw, Poland, 2015.

[21] D. Kim and S. An, "Throughput enhancement by Dynamic Frame Aggregation in multi-rate WLANs," in *Proc. of IEEE SCVT*, Eindhoven, Netherlands, 2012.

[22] S. V. Azhari, O. Gurbuz and O. Ercetin, "QoS based aggregation in high speed IEEE802.11 wireless networks," in *Proc. of IEEE Med-Hoc-Net*, Vilanova i la Geltru, Spain, 2016.

[23] B. Maqhat, M. Dani Baba, R. A. Rahman and A. Saif, "Performance analysis of fair scheduler for A-MSDU aggregation in IEEE802.11n wireless networks," in *Proc. of IEEE ICEESE*, Kuala Lumpur, Malaysia, 2014.

[24] S. Kim and Y.J. Cho, "Adaptive Transmission Opportunity Scheme Based on Delay Bound and Network Load in IEEE 802.11e Wireless LANs," *Journal of Applied Research and Technology*, vol. 11, no. 4, pp. 604–611, 2013.

[25] M. Richart, J. Baliosian, J. Serrat and J. L. Gorricho, "Resource Slicing in Virtual Wireless Networks: A Survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 462–476, 2016.

[26] R. Kokku, R. Mahindra, H. Zhang and S. Rangarajan, "NVS: A Substrate for Virtualizing Wireless Resources in Cellular Networks," *IEEE/ACM Transactions on Networking*, vol. 20, no. 5, pp. 1333–1346, 2012.

[27] S. Zehl, A. Zubow and A. Wolisz, "Hotspot slicer: Slicing virtualized home Wi-Fi networks for air-time guarantee and traffic isolation," in *Proc. of IEEE WoWMoM*, Macau, China, 2017.

[28] M. S. Carmo, S. Jardim, T. de Souza, A. V. Neto, R. Aguiar and D. Corujo, "Towards enhanced connectivity through WLAN slicing," in *Proc. of IEEE WTS*, Chicago, IL, USA, 2017.

[29] J. Saldana, J. Ruiz-Mas and J. Almodvar, "Frame Aggregation in Central Controlled 802.11 WLANs: The Latency Versus Throughput Tradeoff," *IEEE Communications Letters*, vol. 21, no. 11, pp. 2500–2503, 2017.

[30] K. Nakauchi, Y. Shoji and N. Nishinaga, "Airtime-based resource control in wireless LANs for wireless network virtualization," in *Proc. of IEEE ICUFN*, Phuket, Thailand, 2012.

[31] K. Guo, S. Sanadhya and T. Woo, "ViFi: Virtualizing WLAN Using Commodity Hardware," in *Proc. of ACM MobiArch*, Maui, Hawaii, USA, 2014.

[32] M. Richart, J. Baliosian, J. Serrati, J. L. Gorricho, R. Agero, and N. Agoulmine, "Resource allocation for network slicing in WiFi access points," in *Proc. of IEEE CNSM*, Tokyo, Japan, 2017.

[33] E. Coronado, R. Riggio, J. Villalón, and A. Garrido, "Programming Abstractions for Wireless Multicasting in Software-Defined Enterprise WLANs," in *Proc. of IEEE IM*, Lisbon, Portugal, 2017.

[34] D. Xia, J. Hart, and Q. Fu, "Evaluation of the Minstrel rate adaptation algorithm in IEEE 802.11g WLANs," in *Proc. of IEEE ICC*, Budapest, Hungary, 2013.

[35] G. Bianchi, "Performance analysis of the ieee 802.11 distributed coordination function," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 3, pp. 535–547, March 2000.

[36] "OpenvSwitch." [Online]. Available: http://openvswitch.org/

[37] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click Modular Router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, 2000.

[38] "The EmPOWER Protocol." [Online]. Available: https://github.com/5g-empower/5g-empower.github.io/wiki/EmPOWER-Protocol

[39] "Ryu SDN Framework." [Online]. Available: https://osrg.github.io/ryu/

[40] R. Riggio, I. G. B. Yahia, S. Latr, and T. Rasheed, "Scylla: A language for virtual network functions orchestration in enterprise WLANs," in *Proc. of NOMS*, Instabul, Turkey, 2016.