Computer Communications 149 (2020) 176–188

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Ethanol: A Software-Defined Wireless Networking architecture for IEEE 802.11 networks

Henrique Moura *, Alisson R. Alves, Jonas R.A. Borges, Daniel F. Macedo *, Marcos A.M. Vieira *

Computer Science Department – Universidade Federal de Minas Gerais, Brazil

ARTICLE INFO

Keywords:
Wireless networks
Software Defined Wireless Networks

ABSTRACT

Wireless Networks have become ubiquitous to support the growing demand from mobile users, and other devices, like in the Internet of Things (IoT). This article proposes a Software-Defined Wireless Networking architecture specialized in 802.11 Wireless LANs, called *Ethanol*, which provides a more fine-grained control. *Ethanol* is the first wireless SDN architecture that extends the control to the user devices. Further, Ethanol allows intelligent white-box control with finer grain than the state of the art, since it is optimized for WiFi. The proposed architecture is evaluated on a prototype over three use cases. *Ethanol* can be deployed in any Access Point (AP) running embedded Linux, since it has a negligible overhead — up to 1% in memory and 0.1% in CPU usage. Our results show that *Ethanol* dynamically alter the throughput of an application up to 3x during a prioritization period, returning bandwidth to other applications outside this period. Only by controlling the best time to perform the handover, our results show about 45% improvement over the traditional signal-based handover process.

1. Introduction

Wireless networks have become ubiquitous and dense. More devices will become connected to the network because of IoT. Cisco predicts that 50 billion things will connect to the Internet by 2020.[1] Emerging high-speed network standards (e.g. 802.11ad) are migrating to higher frequencies, which are absorbed by walls, thus requiring the densification of APs. Thus, a more refined control of all the network devices is needed to increase future wireless networks scalability in these environments.

Current management architectures for Wireless LANs (WLANs) employ proprietary controllers. These controllers perform network-wide optimizations such as adjustment of transmission power at each AP, selection of best operational channels, faster client mobility, and enhanced traceability, as well as Quality of Service (QoS) policy (rate limiting) enforcement, and security. However, those controllers only manage compatible devices, usually from a single manufacturer, since they rely on proprietary interfaces and Management Information Bases (MIBs). Furthermore, current wireless network controllers for WLAN remain (mostly) closed [1] and the industry has little incentive to change.

This article presents *Ethanol*, an Software Defined Networking (SDN) architecture for IEEE 802.11 WLANs. *Ethanol* refactors the control plane functionalities between the APs, and the controller, creating hooks in the AP implementation that trigger events to be treated by the controller. Also, the controller can use getter/setter methods to change the AP behavior, controlling features such as client mobility, association and disassociation, QoS, link-level parameters and current state, and virtual APs. The key benefit of *Ethanol* is its flexibility, and the use cases it enables, for example those shown in Sections 4 and 5.

The main contributions of our work are: (a) the proposed SDN architecture, which is evaluated in a prototype; (b) a prototype that interacts with the AP and the stations; and (c) we have evaluated the architecture in several uses cases, and this paper shows three original ones. *Ethanol* is an SDN architecture for Wi-Fi networks that runs over the Linux AP implementation, so it can be used on any device that supports IEEE 802.11/2016 protocol. Ethanol focuses on 802.11/2016 protocol (including previous amendments) and, because of that, it provides a more fine-grained control of the APs, e.g., *Ethanol* can configure 802.11ac, and Wi-Fi Multimedia (WMM) specific parameters. We developed a prototype. An evaluation of the demanded hardware resources indicates that *Ethanol* runs well on existing AP with embedded Linux. *Ethanol* is able to request information from the stations, a feature that, to the best of our knowledge, is not present in any other SDN architecture for wireless networks. This is key in managing operations that require interference mapping.

Ethanol was previously published in [2]. Compared to the previous work, this paper provides the following key differences:

* Corresponding authors.
E-mail addresses: henriquemoura@dcc.ufmg.br (H. Moura), alissonralves@dcc.ufmg.br (A.R. Alves), jonasrafael@dcc.ufmg.br (J.R.A. Borges), damacedo@dcc.ufmg.br (D.F. Macedo), mmvieira@dcc.ufmg.br (M.A.M. Vieira).

[1] https://goo.gl/HD7PjN.

https://doi.org/10.1016/j.comcom.2019.10.010
Received 10 June 2019; Received in revised form 2 October 2019; Accepted 3 October 2019
Available online 11 October 2019

- An improved *Ethanol* architecture with a more refined class model, supporting more commands;
- The *Ethanol* protocol focuses now on the control of wireless networks. A Python API was created to implement Open vSwitch Database management protocol (OVSDB), allowing the configuration of Open vSwitch (and other compatible devices), which are not supported by the OpenFlow protocol, and publisher–subscriber features have been incorporated into the implementation to improve the system's response to recurring events;
- *Ethanol* now provides a northbound interface, called *HomeNetRescue* [3], to simplify the development of control applications.

We show three use cases addressing important subjects in wireless networks. The use cases shown in this article demonstrate the versatility of the proposed solution. While a traditional controller is deployed with hardwired generic management programs, *Ethanol* is able to squeeze more performance out of the network due to a white-box design, where the network understands better the requirements, and specificity of the user flows, by interacting with other management systems or with the users' applications. The following use cases are evaluated in this paper:

- Traffic and signal-aware client handover (Section 4.1), which improves the video quality by 45% over a regular handover decision;
- Application awareness for video (Section 4.2), in which CCTV flows are prioritized when relevant events are detected. This allowed 2 to 3 times higher throughput for the video flows;
- The controller's global view of the network allows detection, and remediation of network problems such as mitigating interference in some station's transmission [3] or detecting that a Wi-Fi interface is not transmitting (for example, due to a connection problem between the antenna and the device – Section 4.3)

Our work details the Linux implementation, also discussing the impact of our code in *hostapd* size, as well as CPU and memory consumption. *Ethanol* code is in GitHub under a General Public License (GPL) v2 license. The implementation is open source, based on *hostapd*, and relies whenever possible on the 802.11 amendments, simplifying its deployment. *Ethanol* is now available as a container, e.g. for a serverless architecture. It has been deployed in the FUTEBOL UFMG testbed (http://futebol.dcc.ufmg.br), where any researcher can remotely experiment for free with physical devices.

The remaining of this article is organized as follows. The *Ethanol* architecture and its components are described in Section 2, followed by its implementation in Section 3. As a proof of concept, we develop a prototype of our architecture. Section 4 shows the algorithm of our case studies and the results obtained on a testbed. Section 5 highlights wireless scenarios that can be addressed by *Ethanol*. The related work is discussed in Section 6. Finally, Section 7 presents our conclusions.

2. *Ethanol* architecture

Ethanol is an SDN-based architecture for IEEE 802.11 WLANs with many APs and clients (such as a campus or an enterprise network). This specialization is important to allow our architecture to provide finer grain control over the AP. Besides forwarding, the controller can also control node mobility, authentication, virtual networking, and QoS. *Ethanol* adopts the following design goals: (i) supports IEEE 802.11 as well as Ethernet cards; (ii) does not require changes on the terminals (data collected from terminals relies on existing 802.11 management messages); and (iii) provides APIs for node mobility, AP virtualization, WLAN security, and QoS. *Ethanol* allows the development of custom control software, enabling network managers to run services that fit their needs.

The *Ethanol* architecture has three types of devices: the controller, the OpenFlow-enabled switch and *Ethanol*-enabled APs, as shown in Fig. 1. The controller runs on a computer connected to the infrastructure, i.e, in the wired network or virtualized in the cloud. The *Ethanol* APs are wireless access points that are modified to run *Ethanol* code. We describe *Ethanol* resource requirements later in Section 3 and evaluate them on actual cases studies in Section 4.4.

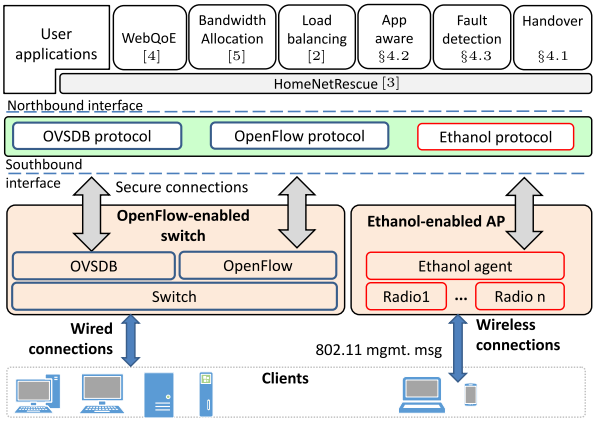


Fig. 1. *Ethanol* AP Implementation.

Ethanol allows the administrator to control network devices, wired and wireless, as shown in Fig. 1. IEEE 802.11 devices are controlled using the Ethanol southbound interface, while switching elements are controlled using OpenFlow and OVSDB. Further, *Ethanol* provides a northbound interface, called *HomeNetRescue* [3], which can be (optionally) used by the network administrator to build management applications. Each component of the architecture is detailed below.

Ethanol devices: It has two components: wired and wireless. The wired device is a configurable switching element that supports the OpenFlow protocol. Since OpenFlow does not provide a control interface for QoS, our solution adds this functionality using the OVSDB defined in RFC 7047. The wireless devices are AP that receive commands from the *Ethanol* controller via a secure channel using a separate connection that handles the *Ethanol* southbound protocol. This approach makes our solution OpenFlow independent, i.e., we can divide the architecture into wireless and wired control protocols, and act independently or in a coordinated fashion on both networks.

Ethanol southbound interface: In order to control APs and user devices, we have proposed a new southbound interface. This interface uses a communication protocol based on two mechanisms:
(i) *Get-Set* model: A controller sends a get or set message to the AP, which reads, modifies or deletes a configuration or wireless parameter. This feature supports a proactive behavior from the controller.
(ii) *Publish–Subscribe* model: The AP offers a set of events to which the controller can register. This supports a reactive, event-based operation.

Northbound interface: Ethanol can be programmed either via its control API, shown in Section 2.1, or one can employ higher-level northbound interfaces. One such example is *HomeNetRescue* [3], which provides an event-based service for the autonomous management of home networks. However, other northbound interfaces could be deployed, for example PANE can be used so the controller interacts with the application servers [4]. We have already developed the following applications over Ethanol:

1. A load-aware handover decision process that improves the quality of experiment in a video application (described in Section 4.1).
2. An application-aware control that dynamically adjusts the application's priority, based on the application events (refer to Section 4.2);
3. A fault-detector, which verifies if the AP is transmitting data into the wireless medium, based on information collected throughout the network (described in Section 4.3);

4. An IEEE 802.11ac network bandwidth allocation algorithm based on the class of user application (video, voice, best effort), providing load balancing in APs [5];
5. A interference mitigation algorithm using Coordinated Transmission Power Control (TPC) among several APs. The algorithm takes into account the interference pattern of each AP [3];
6. A management application that improves the Quality of Experience (QoE) of web clients [6]. The QoE perceived by the user is inferred using machine learning, and the management application tunes the channel and transmission power of the APs;
7. A QoS controller that associates flows to the interface queues in order to prioritize certain wireless flows [2];
8. An intelligent Address Resolution Protocol (ARP) proxy, where the controller solves ARP request from the wired and wireless networks [2];
9. A load balancing application that accounts for the number of clients connected to the AP [2].

2.1. Ethanol control API model

The *Ethanol* API is designed upon an object-oriented approach that works with entities having properties and methods, and handling events. *Ethanol* entities are physical or virtual objects that can be configured or observed. An *Ethanol* AP and a Virtual Access Point (VAP) are examples of a physical and a virtual entity, respectively. Those entities have observable and/or configurable *properties*, such as the number of available channels or Extended Service Set Identifier (ESSID). The properties are accessed by the controller using *get/set* methods or publish/subscribe methods. Finally, entities can have *events*. One such event could be a wireless client requesting an association. The controller registers with the AP which event it wishes to receive a message, and if applicable the threshold for that information. Fig. 2 shows the entities, properties and events of *Ethanol* . To improve readability we have omitted all getter and setter methods, and more information can be found at the documentation in the Github repository. Read only properties are marked with a minus ('−') sign. A filled diamond shape indicates containment, a stronger form of aggregation where the contained objects do not have an existence independent of their container (the class touched by the diamond). Cardinality is represented using Crow's foot notation. All methods with the "ev" prefix correspond to an event that can be managed by the controller.

Some of the proposed properties, methods, and entities may not be feasible on some production APs due to lack of hardware features or limitations on the OS. However, we chose to specify the architecture without taking into account the limitations of existing equipment. For example, some wireless cards do not provide Signal-to-interference-plus-noise ratio (SINR) in their hardware. Our implementation can request this information, and, if the feature is unavailable, returns an error to the controller. Future hardware could be developed with this specification in mind, in a trend similar to what happened with copper SDN switches: the first SDN specifications were limited to functions implementable in existing hardware, and now vendors are proposing chips tailored for SDN operations [7]. Next we provide a brief description of these entities.

2.1.1. AccessPoint entity

This entity represents physical devices. An AccessPoint can have one or more physical radios, represented by the Radio entity, and one or more VAPs running on the AP (VAP class). This entity has three main attributes: $beaconInterval$ (affects the frequency of the beacons), $fastBSSTransition\_compatible$ (if the access point is compatible with fast Basic Service Set (BSS) transition) and $802.11b\_preamble$ (if the preamble is long or short). The methods available in this entity allow the creation and destruction of VAPs, as well as to determine the state of the device. We can retrieve the NICs and the modes they support (e.g. ad hoc, infrastructure), or request an interference map.

2.1.2. Radio entity

This entity configures the physical wireless interface of the AP. It has attributes such as channel, supported bit rates, transmitter power, and allow for power saving mode enable or disable. Radio also gathers link statistics and other information of the wireless radio.

2.1.3. Device entity

This is a superclass to VAP and Station entities. It implements common functionality to configure and collect information from the device. This entity contains information such as the MAC and IP addresses, and if the station supports 802.11 QoS modes. The device entity also collects information about the link between the station and the AP, such as number of bytes/packets received and sent, signal strength, SNR, bitrate, number of retries, and packet loss.

2.1.4. VAP entity

A physical device can have zero or more VAPs (a VAP can be configure but kept disabled for future use or fast startup). Stations connect to a VAP, and a group of VAPs form a Network. Note that a VAP represents a network inside a physical device, so several users can connect to one VAP running in one physical device. A VAP does not broadcast its Service Set Identifier (SSID) if $broadcastSSID$ is disabled. Also VAP controls MAC transmission parameters such as guard and DTIM intervals, RTS threshold, and link capabilities (e.g. if frame burst is enabled). It also exposes the contention parameters of 802.11 QoS BSS (e.g. maximum and minimum contention window values, AIFS values) and admission control parameters. Each VAP also has its own security parameters. VAP inherit the properties of the Device superclass. User association and authentication generate events in the controller, which may allow or deny the request. The entity also has events to respond to fast transition and fast reassociation, as defined in IEEE 802.11/2012 BSS Transition Management, or for probe requests.

2.1.5. Network entity

A network may contain several VAPs. This entity represents the network and its SSID. It provides methods for the association and dissociation of APs with the network, as well as a method to request a user handoff with 802.11 fast transition, if supported.

2.1.6. Station entity

This entity represents a user station, and inherits properties and methods from the Device superclass. Wireless metrics, e.g. bytes received or bytes sent, is collected using messages from existing 802.11 standards. This entity also returns measurements collected using 802.11 radio resource management. Examples are channel reports ($getLoadInfo$, $getNoiseInfo$, $getInterferenceMap$), a list of APs in range ($getAPsInRange$ — useful for pre-handoff optimizations), counter group values (e.g. transmitted fragment counts, multicast transmitted frame counts, and failed/retry counts) using $getStatistics$ method, among others.

2.2. Contributions and innovations

Ethanol is an enabler for intelligent wireless networks, since it allows control algorithms to monitor the network's parameters, and then actuate over the network. Having such a substrate, an intelligent control loop has finer control over the wireless links. One example of such an intelligent controller can be found in [6], where the sensing of the wireless parameters in *Ethanol* AP improves the quality of experience of web browsing applications. The control uses a machine learning approach to learn the best configuration for each state of the network, and sends commands using *Ethanol* API.

Many proposals provide architectures capable of managing various types of wireless networks, e.g. EmPOWER works with 5G and Wi-Fi. *Ethanol*'s main differential is the higher number of controllable parameters when compared to generic architectures such as EmPOWER. Since

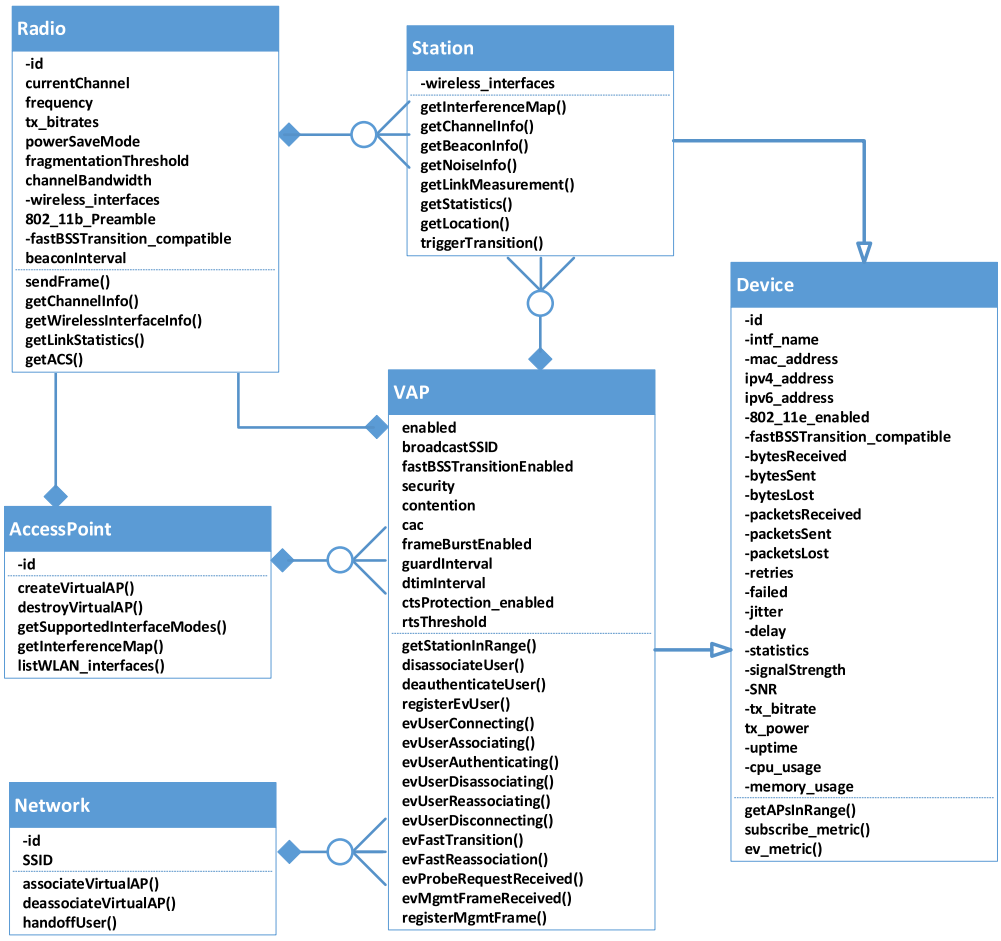


Fig. 2. *Ethanol* control API model.

Ethanol is specific to networks compatible with the IEEE 802.11 standard, it provides the network admin with a larger set of commands and events that can be explored in intelligent control loops. The drawback is that *Ethanol* does not control multiple Radio Access Technologies (RATs). However, because its open API, it can be used inside a broader RAT controller to deal with the IEEE 802.11-compatible devices.

Corporate IT departments have to deal with a large number of APs whose transmission areas overlap. IEEE 802.11 does not provide automatic channel selection or message exchange mechanisms to reduce interference caused in these dense environments. Companies therefore turn to commercial controllers, which use proprietary mechanisms to reduce co-channel interference [8]. *Ethanol* allows the creation of mechanisms for neighboring APs to cooperate and operate on different RF channels [3]. Moreover, although in a dense network the stations have several possible AP options, this degree of freedom is not fully exploited in the IEEE 802.11 protocol. This is because standard Wi-Fi stations select which AP to associate with using pure local information, e.g. the signal strength. This option is suboptimal, especially in non-homogeneous scenarios where BSS may have a varied number of stations with diverse demands. Slower stations also monopolize airtime and thus significantly decrease network capacity [9]. In addition, usually once a station is associated, it remains connected to the same AP, even if there is another AP capable of providing better QoS, for example, higher link quality or lower utilization. *Ethanol* can improve these scenarios by managing the bandwidth as shown in [5], or stirring the station to a better AP as shown in [2], and in Section 4.1.

These are points where *Ethanol*'s contribution is important because: (i) The use of a programmable, open architecture that covers many of the features of IEEE 802.11 APs allows current devices to perform new tasks with logically centralized coordination; (ii) The global view provided by the controller allows the Wi-Fi network to handle a higher density of APs, as it allows mitigation of interference [3]; (iii) Because of the high number of devices to be managed, it is necessary to perform automatic and smart management, for example by creating automatic problem management platforms [3], to provide dynamic configuration of channels [5], etc. One way to accomplish this is to use learning mechanisms that allow the controller to react to changes in the network aiming, for example, to improve the user satisfaction [6]; and (iv) To provide fine grain QoE, the network should be application-aware. We show in this paper that *Ethanol* enables the administrator to build control modules over *Ethanol* that can interact with other systems applications.

Finally, *Ethanol* is the first SDN architecture, as far as we know from the state of that art, that extends the SDN control up to the user devices. As we will show in the use cases in Section 4, the monitoring and control of the user devices allow significant improvements in the operation of Wi-Fi networks. Although cellular networks expose configuration parameters of the user equipment to the administrator for quite some time, this has been neglected by existing wireless SDN architectures.

3. Ethanol implementation

We implemented an *Ethanol* prototype in order to evaluate the basic functions of the architecture. Due to time restrictions, our prototype was implemented using a Linux computer. However, our implementation can be deployed into commercial routers running Linux-based OS. The implementation in this article is an improvement over the code presented and evaluated in [2]. The *Ethanol* source code is available

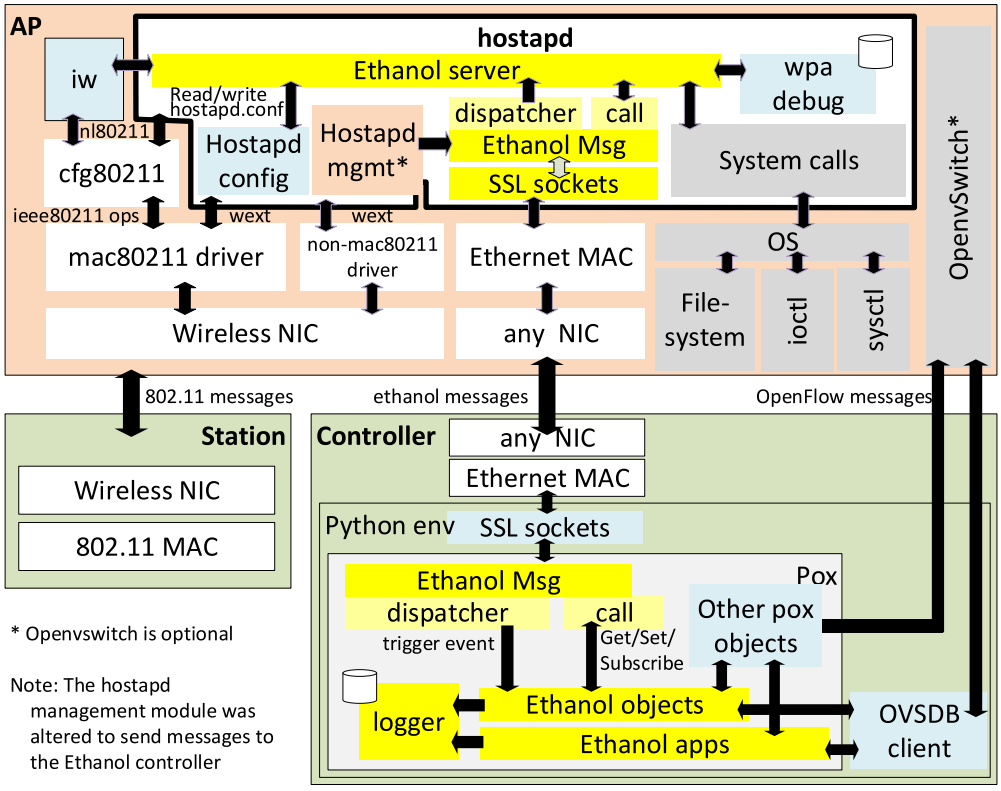


Fig. 3. *Ethanol* component model. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

as open source in GitHub.[2] *Ethanol* can also be downloaded as a docker container. Further, any researcher with a free Fed4Fire account (the European federation for experimental testbeds) can experiment remotely with *Ethanol* in the FUTEBOL UFMG testbed.

Fig. 3 shows *Ethanol* component structure. *Ethanol* was implemented on top of *hostapd*, and uses SSL Sockets to secure the connection to the controller. On the APs, we installed OpenvSwitch version 1.1+ (http://openvswitch.org/). OpenvSwitch is an optional component, and without it *Ethanol* controls only the wireless network interfaces.

The *hostapd* code has been modified to run an Ethanol agent. The modification are shown in yellow in Fig. 3. The main modifications are: (i) *hostapd* management functions have been changed to add calls to the *Ethanol* controller; (ii) The *Ethanol server* module was added to *hostapd* to handle the specific wireless network messages. This module interacts with the messaging module to receive control messages from the controller, and to send response to requests or to programmed events; (iii) The log messages are generated using *hostapd*'s *wpa_debug* module, following the general *hostapd* log configuration. *Ethanol* can also change the "hostapd.conf" file using remote commands sent by the controller. Our code has low impact on *hostapd*'s final size (less than 1%) as shown in Table 1.

The *Ethanol* controller was implemented as a POX module. Fig. 3 shows in the box named "Controller" the controller implementation. The communication is made by any network interface that has a route to the desired destination, allowing for both in-band or out-of-band management. Although POX is an outdated controller with partial support to OpenFlow 1.3, we chose it since it is suited for fast prototyping, due to the wealth of documentation and developer forums, and also because of the simplicity to program complex code in Python. In future work, we intend to migrate our controller to Ryu (https://osrg.github.io/ryu/), since it is nowadays more mature than POX, fully supporting OpenFlow version 1.5.

[2] https://github.com/h3dema/ethanol_hostapd for the AP, and https://github.com/h3dema/ethanol_controller for the controller.

We note that in the proposed architecture there is no station-specific module, so the station control uses only IEEE 802.11 standard management features. However, for some of our cases studies, the functionality provided by IEEE 802.11 was not available at the client, so in these cases an agent that implements the required functionality was installed in the station, e.g., in the handover use case (Section 4.1) an agent implements the Link Report message.

4. Case studies

This section describes three experiments performed with the prototype in this paper. We show in the following subsections that: (i) *Ethanol* provides better user mobility control. (ii) wireless traffic can be prioritized over wired traffic, and the network can be made aware of application needs; (iii) *Ethanol* can detect faulty APs. In our previous work [2], we showed three different cases studies. These cases show the architecture's ability to use the OpenFlow and *Ethanol* features to improve the quality of traffic in the wireless network using flow prioritization and using ARP filtering, and also that *Ethanol* can load balance the APs. Other uses of Ethanol have already been published in subsequent papers from our group [3,5,6].

In the experiments, each AP runs on a PC with one Intel Dual Core 2.5 GHz processor, 2 GB of RAM, one 1 Gbps Ethernet PCI card and one Atheros 802.11bgn PCI card. The controller runs on a virtual machine with two Xeon Core 2.2 GHz processors, 4 GB of RAM, and one virtual 1 Gbps NIC. The mobile stations are ASUS notebooks with an Atheros 802.11bgn card. The controller, AP and mobile station run Ubuntu LTS 14.04. The end of this section shows a performance analysis for the AP and the controller in each use case to identify the impact of our implementation on the CPU and the memory usage.

4.1. Load-aware handover

Machań and Wozniak [10] highlight that in wireless networks handover performance is critical to support multimedia services. In this

Table 1
Impact of our modifications on *hostapd* size.

Software		Size (in bytes)	Number of lines w.o. comments	
			Code	Header
Original *hostapd* code		3,916,595	198,144	20,142
Ethanol functions and message module	(+)	23,724	22,406	4,340
Modified *hostapd* code	(=)	3,940,319	220,550	24,482

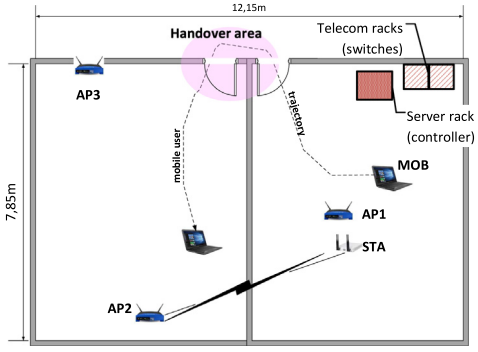


Fig. 4. Handover experiment. Dotted line shows mobile user trajectory.

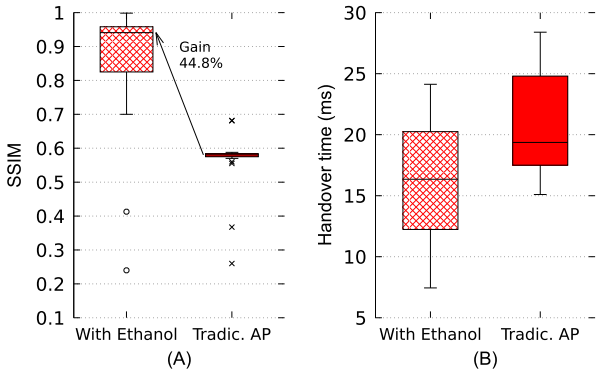


Fig. 5. Boxplot showing the distribution of: (A) SSIM and (B) total handover time (ms) for architectures with and without Ethanol (traditional IEEE 802.11 AP). Ethanol improves SSIM by 44.3% and reduces median handover latency by 4.96 ms.

scenario we propose and evaluate a traffic-aware handover algorithm, showing that the quality of a video stream improves during the handover when the load-aware algorithm is active.

In IEEE 802.11 the client selects the optimal AP to connect to, and this choice is left for the NIC vendor. Typically the client selects the AP with the highest Received Signal Strength (RSS), neglecting the unused capacity of each AP. Disregarding the current load of the AP may lead to an uneven load distribution across the network. An SDN approach can take load as well as other factors in consideration, improving the performance.

The proposed solution uses the "measurement report" defined in the 802.11k amendment so that the controller via AP requests the signal information from the station. However, this feature is implemented only by a few vendors, e.g. in Apple's iPhone. Since our station does not have this feature, an agent running on the station provides this functionality. This agent also allows the controller to set a threshold for the SNR and it sends a message to the controller (via SSL socket) when this threshold is exceeded. In [11] the authors create a RSSI trigger, but their metric captures the AP side of the connection, while ours uses the client side.

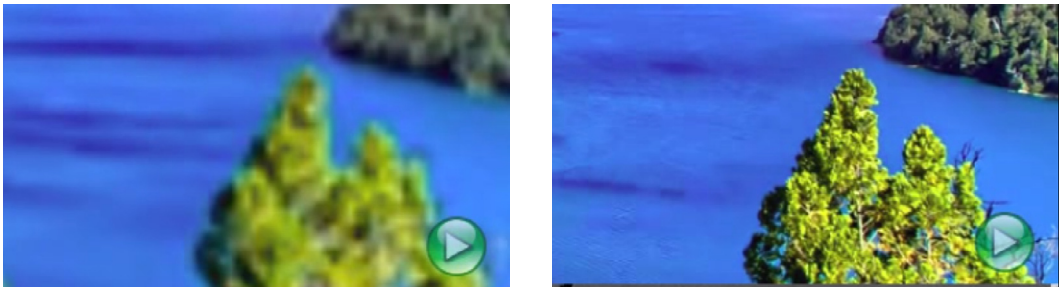
Algorithm 1 shows the handover decision process. When a Signal-to-noise ratio (SNR) threshold is reached at the mobile station (MOB), it sends a message to the controller, triggering the algorithm. We could have used RSS instead. The controller inquires the station about which APs are in its range (line 2). From this list, the algorithm selects only those APs controlled by *Ethanol* (line 4) that satisfy some signal threshold (line 6). This list of APs is sorted by traffic, and the AP with less traffic is returned. If the list is empty, the MOB remains in the same AP.

The controller sends to the station a BSS transition management message, as defined in IEEE 802.11v amendment, and incorporated in 2012 to the main standard text. This message requests the station to transition to a specific AP, selected by the controller. The station then roams to the new AP following the fast BSS transition process defined in IEEE 802.11r, also incorporated in the main body in 2012. So a fully compliant station does not require any modification to work with the proposed implementation.

Algorithm 1 Handover

```
1: function SNR_THRESHOLD_REACHED(sta, ethanol_aps, τ)
2:     inRange ← sta.getAPsInRange()                    ▷ get APs detected by the station
3:                                  ▷ all aps in range that belong to Ethanol, except current AP
4:     candidates ← ethanol_aps − inRange ∩ {sta.connectedTo()}
5:                                  ▷ select only AP with SNR greater than threshold
6:     aps ← filter(apifap.vap[0].SNR > τ)
7:                                  ▷ return the AP with less traffic, if it exists
8:     return sortedByTraffic(aps, sort = DESC)[0]
```

Our handover approach is distinct from BigAP [12], because BigAP exploits the DFS capability to announce channel switches. The AP sends a channel switch message informing the station that it will go to channel x, but the AP does not change channels. Another AP, already in



(a) without *Ethanol* (b) with *Ethanol*

Fig. 6. Video quality during handover.

channel x, associates the station, and performs the handover operation. This approach can also be implemented using *Ethanol* as future work. Our handover approach is also distinct from [11], which benefits from the Lightweight Virtual AP (LVAP) abstraction to transfer the client from one physical device to another, while ours rely on IEEE 802.11 standard messages to do so. We can also use *Ethanol* to implement the approaches in [13], and [14].

Evaluation. To demonstrate the capability described in the first part of this section, we have created the scenario shown in Fig. 4 where there are three *Ethanol*-enabled APs (AP_1, AP_2 and AP_3), configured to provide the same wireless network (ESSID) in the same mobility domain. APs may be configured on the same or different channels, provided the station is able to detect them. The APs are in the same mobility domain, using 2.4 GHz bands. AP_1 is using one channel, say $channel_1$, while the other APs are using another channel, $channel_2$. These APs are connected to the Ethernet network via a gigabit switch. A video streaming server is also connected to this switch. The wireless station (MOB) is initially connected to AP_1 and starts downloading a video stream from the server. There is a stationary station (STA) that is connected to AP_2. This station also downloads a stream from the video server. MOB moves in the direction of AP_2 and AP_3, as shown in Fig. 4, thus the signal of AP_1 becomes weaker, and the signals from AP_2 and AP_3 become stronger. The transmission power in these APs is set to 15 dBm, 10 dBm, and 9 dBm, respectively. The signal from AP_2 is stronger than that AP_3, however AP_3 has no traffic. We perform two experiments: (1) without the Ethanol controller, and therefore the station switches to AP_2 (better RSS) when the AP_1 signal becomes weaker enough; and (2) the Ethanol controller interferes in the handover process, considers the traffic in the APs network interfaces, and decides to switch to the AP that is less overloaded (in our case AP_3). Ten repetitions were performed for each network configuration.

We used Video Quality Measurement Tool — VQMT, to analyze the image quality of the videos. A Docker image is available at https://github.com/h3dema/ubuntu-vqmt. To infer the QoE, we use the Structural Similarity (SSIM) index, which measures the quality of digital television [15]. SSIM measures the structural distortion of the video, aims to have a better correlation with the subjective impression of the user, and provides output values between 0 and 1. The closer to 1, the better is the quality of the video [16]. As SSIM needs a reference video for comparison, we use the original video.

Results. Fig. 5(A) shows the SSIM values for the scenario without *Ethanol* and with *Ethanol* running. The average SSIM is 44.8% higher when using Ethanol, since the distances between the APs are small, and the traffic is diverted to the AP with less load. We run a two-sample t-test of equal means with a 95% confidence interval, considering the variances equal but unknown. This test rejects the null hypothesis, meaning that the SSIM values are different.

Fig. 5(B) shows the total handover time, measured at the mobile station, in milliseconds. The right boxplot shows results using only the IEEE 802.11 fast BSS transition in a mobility domain. The other boxplot shows the results using *Ethanol*. The handover without *Ethanol* (traditional 802.11 AP) lasts on average 20.84 ms, and the handover using *Ethanol* lasts on average 15.88 ms, reducing the average time in 23.8%. However this difference is statistically not significant. We run a two-sample t-test of equal means with a 95% confidence interval, which indicates that our modifications in APs do not affect the average handover time. This is expected because our handover relies on IEEE 802.11 handover process implemented in *hostapd*.

Using *Ethanol* improves the video quality perceived by the user during the handover, as can be seen in the excerpt of a video frame, as shown in Fig. 6. The *Ethanol* image (6b) is sharper, while in the unoptimized handover the landscape looks blurry.

4.2. Adjusting network capabilities based on application needs

The *Ethanol* controller can interact with an application, adapting the network to the needs of the applications, since Ethanol has an open and programmable interface.

In this use case the *Ethanol* controller interacts with an object detection service of a CCTV system using simple SSL socket messages. A camera with low processing capacity (e.g. an Wi-Fi security camera) is connected to the wireless network. The images captured are sent to an image processing server ($ImgSrv$), which detects an object of interest in a particular detection area, for example, a person in a restricted area. The server does object recognition using OpenCV libraries (http://opencv.org/). When $ImgSrv$ detects a person's presence, it informs the *Ethanol* controller that the bandwidth allocated for the camera needs to be increased, so that the tracking can be more effective (i.e. more frames per second will be captured). $ImgSrv$ also requests the camera to increase the amount of frames per second. $ImgSrv$ communicates with the controller using an SSL socket. It sends a flag message (detected/not detected) so that the controller knows when to increase the allocated bandwidth.

We employ WMM [17] to prioritize traffic, and hence increase the bandwidth of the camera. When the system boots, *Ethanol* classifies all traffic as "best effort" (AC_BE). When the $ImgSrv$ detects a person, the *Ethanol* controller reclassifies the traffic from the camera to a higher priority (AC_VI). When $ImgSrv$ no longer detects a person in the area of interest, it informs the *Ethanol* controller, which downgrades the traffic to AC_BE.

According to WMM, it is advisable to change the classification of the frames on both sides of the link in order to ensure priority in the uplink and the downlink. Thus, the best option is to install an agent in the client that marks the station's outgoing packets with the correct priority according to the $ImgSrv$ decision. To avoid changing the implementation in the client, we use a little trick. The camera is connected to a unique SSID and all video frames it transmits are marked as AC_BE. However the controller, when determining the camera's priority, does three actions in the AP: (1) it creates packet rewriting rules (both input and output) to that particular camera; (2) it changes the parameters related to AC_VI to correspond, as appropriate, to a normal priority or a video priority; and (3) it sends a "QoS Map Configure" message transmitted from the AP to the station, as defined in IEEE 802.11u, and incorporated to the main standard in 2012, which provides a Differentiated services code point (DSCP) mapping to User Priorities (UPs) used in WMM. In this way we managed to deceive the station that despite considering all traffic as best effort, the station maps to UP = 4 (AC_VI) or UP = 0 (AC_BE).

This kind of dynamic classification is not possible with current APs because they are not aware of the demands of the application.

Evaluation. Fig. 7 shows the experimental setup. $VideoCapture_1$ is connected to the network using only its wireless interface, and has the same hardware configuration as the AP plus a Logitech USB camera. For the experiment, a person walks in the room, being sometimes detected by the camera, as shown in Fig. 7. To show the effect on traffic, we also connect another wireless station ($Download_1$) that simulates a download during the experiment lifetime. This traffic is always classified as "best effort", and it is generated using *iperf* (https://iperf.fr/) in UDP mode. AP_1, the image processing server $ImgSrv$, and the *Ethanol* controller are connected to an Ethernet network.

Results

Fig. 8 shows the throughput for $Download_1$ and for $VideoCapture_1$ during the experiment. The two gray areas in the figure highlight when a person is detected and the priority of the camera flow is increased. We notice in these areas that $VideoCapture_1$ traffic increases 2x to 3x due to the increase of the amount of frames transmitted, and at the same time the $Download_1$ traffic is reduced due to the higher priority of the video traffic.

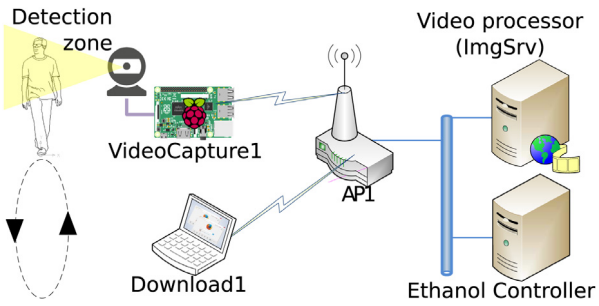


Fig. 7. Application-aware experiment setup.

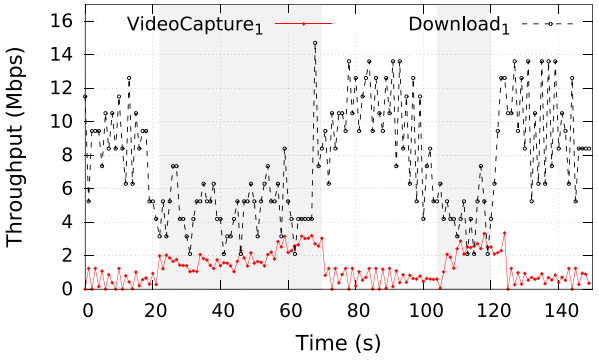


Fig. 8. Throughput for the application-aware management scenario.

Algorithm 2 Detecting an AP's Faulty Wi-Fi Interface

```
1: function evBeaconFrameReceived(EthanolAP, DetectedAP) ▷ handles beacons detected by an Ethanol AP
2:     inRange[DetectedAP] ← currentTime()  ▷ place the detected AP in the "in range" list
3:
4:                                          ▷ function executed each (VERIFY_TIME) seconds
5: function detectFaultyAps(inRange)
6:                                          ▷ set N corresponds to all detectable APs
7:     N ← AccessPoint.getVAPs()
8:                                          ▷ set N' represents all Ethanol detected APs
9:     N' ← inRange_{currTime−EXPIRATION_TIME} − inRange_{currTime}
10:    return N ∩ N'                        ▷ resulting set corresponds to faulty APs
```

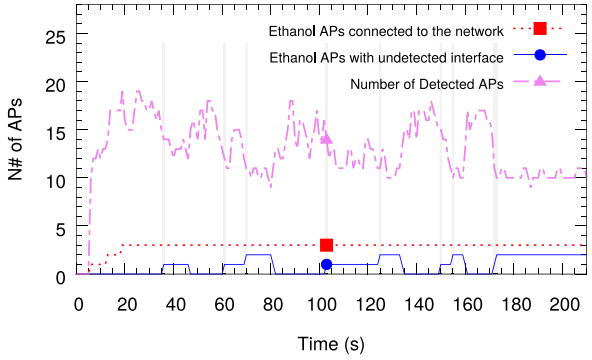


Fig. 9. Detecting faulty Wi-Fi interface. The solid blue line shows the number of *Ethanol* APs that are not detected, and the dashed one shows how many Ethanol APs are running at the time.

Although in this scenario we only modified the priority of the flows, a number of other wireless parameters could be adjusted depending on the application's needs. For example, in [6] we adjusted the transmission power and the channel in order to improve the QoE of a Web application. Those adaptations would not be possible on existing wireless controllers, since there is no way to embed application awareness into the pre-installed control loops. However a traditional IEEE 802.11 AP cannot dynamically allocate bandwidth to the stations, which in our case represent a loss of up to 66% in the desired throughput.

4.3. Detecting faulty Wi-Fi interfaces

Nowadays, SDN architectures such as OpenFlow are able to identify link down events only on the wired network. By combining data from the wireless network, the administrator is able to pinpoint other sources of failure in WLANs, such as a weak signal, damaged Ethernet interface, etc.

Network information can be obtained with *Ethanol* using the IEEE 802.11k amendment.[3] With this protocol, the controller can request the list of APs that a station or an AP can contact by performing active or passive scans. If IEEE 802.11k is not implemented on the stations, the controller can program its APs so they periodically scan the wireless medium. With these scans, it can identify whether an AP's network interface stopped transmitting or a rogue AP was in range. If an AP_x no longer detects AP_y's beacons, one can infer that its wireless interface has failed, since this AP_y is still reachable by the controller via Ethernet. Algorithm 2 shows this detection algorithm.

The controller receives a list of APs within reach of an *Ethanol* AP. The controller reacts to an evMgmtFrameReceived() event, triggered when an AP receives a beacon, calling the function on line 1. It receives two parameters: the hardware address of *Ethanol* AP that detected an AP, and the hardware address of the detected AP. This tuple is stored in *inRange*. Each *inRange* entry is timestamped. Line 2 updates the detected AP timestamp entry. This tag controls the expiration time of the entry, and can be changed using a configuration parameter called EXPIRATION_TIME.

The DetectFaultyAps procedure runs every VERIFY_TIME seconds. *AccessPoint.getVAPs*() returns a set of *Ethanol* APs that are connected to the controller, as shown in Line 7. Line 9 determines all APs that can be detected by the *Ethanol* APs, i.e APs with transmitting wireless interfaces. Then, in Line 10, the algorithm returns the set of non-detected *Ethanol* APs, therefore they have a wireless problem.

Note that this procedure only works for APs that are in range of other *Ethanol* APs or stations connected to those APs. So at least one AP should be working to collect data from the wireless medium.

Results. To test the detection algorithm described above, we set up an experiment with 3 *Ethanol* APs. These AP are connected to the controller via Ethernet. During the experiments, the AP's antennas were manually disconnected and reconnected.

Fig. 9 shows an experiment run, where the Y-axis show the number of APs connected to the controller, and detected in the neighborhood. In the first 20 s, the APs are turned on one by one. They connect to the controller, as shown by the dashed line in Fig. 9. The dot-dashed line, named "Number of Detected APs", shows how many APs are detected in the environment, including non-*Ethanol* APs. During initialization, an AP performs two processes: (1) it connects to the controller using OpenFlow protocol, and (2) it sends a Hello message to the same controller using *Ethanol* protocol.

We realize that the AP's wireless interfaces are working because the "Ethanol Routers connected" line shows a non-null value, and "Ethanol Routers with undetected interface" line is zero. Failures are shown in Fig. 9 by the line "Ethanol Routers with undetected interface". A failure is created by disconnecting the AP's external antennas. To recover from the failure, the antennas are manually reconnected.

In Fig. 9, a vertical gray bar in the background indicates the antenna disconnection. This gray bar represents the ground truth. Notice that

[3] Incorporated in the main body of the standard in 2012.

Table 2
Controller and AP relative memory and CPU Usage.

Use case	Usage (%)			
	Controller		AP	
	Memory	CPU	Memory	CPU
Detecting Faulty Wi-Fi Interface	1.53 ± 0.07	**7.86** ± 0.3	**1.24** ± 0.02	1.195 ± 0.028
Application-aware network capacity	0.55 ± 0.0058	0.915 ± 0.0485	0.16 ± 0.0013	0.08 ± 0.0046
Handover control	5.09 ± 0.0052	0.033 ± 0.005	0.04 ± 0.0038	0.096 ± 0.0108

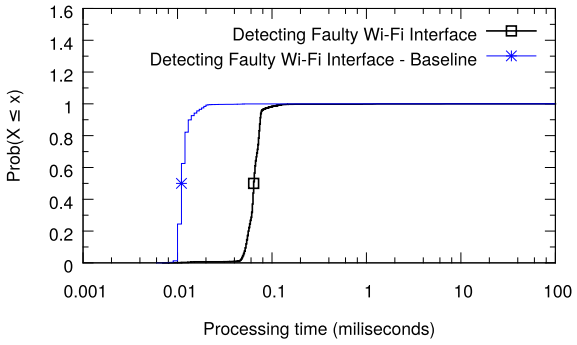


Fig. 10. Impact on Hostapd's processing time in Detecting Faulty Interface Experiment.

whenever the blue line goes up, it means that an AP had its antennas disconnected and this event is detected by the controller using Algorithm 2. If the blue line goes down, it means that the antennas have been reconnected and the controller detects the presence of Ethanol AP among the APs detected in the wireless medium. The change in Y-axis indicates how many APs have been disconnected or connected. We see in $X = 36$ s that the controller identifies one AP with a faulty Wi-Fi interface. In $X = 46$ s, this AP's antennas are reconnected, and the "Ethanol Routers with undetected interface" line goes back to zero. We repeat the procedure, disconnecting another AP's antennas, in $X = 61$ s and $X = 71$ s. After that the line shows two APs that are not connected. After that, we reconnect the antennas, in $X = 82$ s. This way, we observe that the controller is able to detect problems in the wireless interfaces.

Fig. 10 shows the cumulative distribution of the execution time of the management function in *hostapd*. The baseline is the time necessary to process a beacon in *hostapd*, while the other curve shows the time it takes to run these function, while communicating the information to the controller and acting according to the controller's decision. The increased execution time shown in Fig. 10 is compatible with the overtime necessary to send a message to and from the controller. The X-axis is in logarithmic scale to highlight the difference of both curves. The execution time without threads (not shown in the Figure) is about 6.46 ms for 95% of our measures. Using threads improves the performance because beacons are very frequent. The execution time for the threaded implementation is about 77 µs for 95% of our measures. The execution time for *hostapd* procedure without our implementation, i.e. without sending beacons to the controller, is 16 µs. The difference from our threaded implementation and the unmodified *hostapd* implementation is explained by the overload caused by message and thread creation and execution.

Kerravala [18] showed that Wi-Fi related issues are more commonly reported compared to other network issues. Thus detecting faults is an important administrative task. This section shows that *Ethanol* is capable of detecting network failures, which is an important factor for solving the main problem reported in [18]. This topic is not covered by IEEE 802.11, so companies turn to proprietary solutions that typically provide only an indication of a non-responding device. With *Ethanol*, it is possible to create an open architecture for troubleshooting, as presented in [3].

4.4. Evaluation of CPU and memory usage

This section evaluates the CPU load and memory usage of the wireless libraries in the controller and in the *Ethanol* AP using *sar* from the *sysstat* package.[4] The objective is to understand the CPU and memory usage, to understand whether *Ethanol* can run on commercial wireless AP. Although in this evaluation the Ethanol AP is running on a Linux PC, we expect that the overhead of our implementation will be similar in APs using embedded processors. This is because many commercial APs run Linux, using the same *hostapd* implementation that is installed on a PC. Hence, instead of showing the amount of CPU and memory consumed by the modified libraries, we opted to show the percentage of increase compared to the unmodified libraries.

Controller. Table 2 shows the CPU load, and the memory used by user processes, both in percentage, which were consumed by the experiment with *Ethanol*. It also shows the 95% confidence intervals for the both metrics. Notice that the confidence intervals, especially those related to memory usage, are very small. We considered the means independent, and run a two-sample t-test of equal means with a 95% confidence interval. The test rejected the null hypothesis, hence the means are different.

Further, we compare the *Ethanol* implementation with a baseline. This baseline is collect running only the OS in the controller hardware, with the POX controller, the DNS and DHCP programs deactivated. The maximum increase due to Ethanol is about 8% in CPU and 5% in memory usage. The fault detection experiment shows the highest processor usage among the case studies. This behavior occurs because the program in the controller is called more frequently, as every beacon detected by the APs triggers a call. The application-aware experiment imposes small load to the controller, less than 1% in CPU, and about 0.5% in memory. It only needs temporary memory for local variables, and for *Ethanol* objects. The controller in the handover control experiment has low impact on CPU, and around 5% impact on memory usage. This behavior is expected since the controller is demanded only when the station reaches the threshold, even then it generates a peak of less than 1% CPU use. However when triggered, the controller collects a large amount of neighborhood information that is stored temporarily in controller memory.

AP. Table 2 presents the impact measured in the APs on both metrics – CPU and memory usage – relative to a baseline, which is obtained running *hostapd* without *Ethanol*'s modifications. We see that the values do not generate a significant burden to the hardware, because the case that consumes more CPU uses only an extra 1.2% of the total memory. The values for the AP are less scattered than the ones we measure in the controller — smaller confidence intervals. We also observe that fault detection and association control experiments use more processor than the others. There is almost no impact in the application-aware experiment — less than 0.2% of memory and 0.1% of CPU increase. This is expected because the only interaction with the AP is the changing of priority rules. Our implementation in the handover control experiment uses almost no extra memory, and also has low CPU footprint as shown in the table.

[4] http://sebastien.godard.pagesperso-orange.fr/.

Table 3
Comparison between proposals/products.

Articles/proposals	AAA	QoS	MOBILITY	MONITOR	FORWARDING	NETWORK	MODIFICATION
Commercial controllers [8,19–21]	Y	Y	Y	EXCL/PARCIAL	COURSE	WiMax, WLAN	AP
BeHop [22]	Y	N	N	NON-EXCL/PARCIAL	Y	WLAN	AP
BigAP [12]	N	N	Y	NON-EXCL/PARCIAL	N	WLAN	AP
CAPWAP	Y	Y	Y	-	FINE	WLAN	AP
CloudIQ [23]	N	Y	Y	-	FINE	3GPP LTE	BS
CloudMAC [24]	Y	Y	Y	POSSIBLE	FINE	WLAN	AP
CoAP [25]	N	N	N	NON-EXCL/PARCIAL	NO	WLAN	AP
CROWD [26]	Y	Y	Y	-	FINE	3GPP LTE, WLAN	BS, AP
EmPOWER [11,27]	N	N	Y	NON-EXCL/PARCIAL	FINE	3GPP LTE, WLAN	BS,AP
Jigsaw [9]	N	N	-	EXC-TOT	-	WLAN	AP
Ætherflow [28]	Y	Y	Y	NON-EXCL/PARCIAL	Y	WLAN	AP
MobileFlow [29]	N	Y	Y	-	FINE	3GPP LTE	BS
Odin [30]	Y	-	Y	-	-	WLAN	AP
One Big AP [31]	Y	N	Y	NON-EXCL/PARCIAL	-	WLAN	AP
OpenRF [32]	N	Y	-	-	-	WLAN	AP
OpenRoads [1]	Y	-	-	-	-	WiMax WLAN	AP
OpenSDWN [33]	Y	Y	Y	NON-EXCL/PARCIAL	FINE	WLAN	AP, Middlebox
SDWLAN [34]	Y	N	Y	NON-EXCL/PARCIAL	COURSE	WLAN	AP
SoftCell [35]	N	Y	Y	-	FINE	LTE	BS
SoftRAN [36]	Y	-	Y	-	-	LTE	AP
SWAN [37]	Y	N	Y	NON-EXCL/PARCIAL	COURSE	WLAN	AP
Ethanol	Y	Y	Y	NON-EXCL/PARCIAL	FINE	Ethernet, WLAN	AP

Note: CAPWAP and CloudMAC concepts can be applied to any wireless network however in the article they use IEEE 802.11.

5. Other uses of Ethanol

This section presents wireless networking problems that can be addressed using *Ethanol*, in addition to the case studies presented in Section 4.

Ethanol can be used for transmission power control. *HomeNetRescue* [3] profited from our architecture to create a control solution that mitigates interference with coordinated TPC, improving the throughput by 66% and the delay by 36% when compared to a network without power control.

The transmission characteristics may vary for every transmitted packet in wireless networks. *Ethanol* architecture provides methods to retrieve these characteristics. *Ethanol* can retrieve the vicinity of a client node so it dynamically reports available radio resources. In an 802.11/2012 enabled network, clients and APs can send reports to each other (neighbor, beacon, and link measurement), allowing the controller to have a better understanding of the wireless medium. In [11], the authors show the importance of obtaining the interference map to improve channel quality. Alves et al. [3] use *Ethanol* to improve throughput by selecting the best channel considering also the interference. An application that orchestrates wireless channel assignment in a multi-AP environment using *Ethanol* was shown in [3], while a bandwidth dynamic allocation in a chosen channel was shown in [5].

Ethanol uses IEEE 802.11 security features, but can interact with the network devices to detect unauthorized APs, since those may degrade the performance of official APs and expose the network to unwanted access. The detection of rogue APs requires the analysis of the wireless medium in different points of the network. This is feasible using *Ethanol*, using APs or even client stations. *Ethanol* can be used to redirect, filter or drop rogue traffic directly in the APs, thus reducing the overload on the wired enterprise network.

Localization in a wireless network can be used for handover decisions, or to identify nodes in case of security breaches or malfunctioning devices. *Ethanol* provides tools to perform RF characterization using 802.11k messages or the controller can run indoor localization algorithms (e.g. [38]) or use Wi-Fi fingerprinting on dense deployments [39]. Finally, *Ethanol* can access the beamforming matrix on IEEE 802.11ac/ah devices, which can be used to compose a user localization vector [40].

Serverless computing is becoming increasingly relevant [41]. *Ethanol* is now also provided as a Docker container [42], and exposes a management interface, thus it can be used as a microservice in serverless computing platform. A new application will be developed to communicate with OpenLambda [43] as a future work.

6. Related work

SDN paradigm separates the control plane from the data plane, allowing to dynamically program the network [44]. It reduces the complexity of the network management, and promotes innovation. However, in OpenFlow-based SDNs, the data plane is only programmable on switching elements. We advocate that SDN-enabled APs will be able to improve the management of WLANs even further than commercial controllers. An open API enables the deployment of context and application aware control algorithms, an impossible task on current wireless controllers.

As SDN has generated many innovative applications on the wired domain, there are also many proposals on wireless networks. Table 3 shows a comparison of the proposals presented in this section. If the proposal considers aspects of the column, we mark it with a **Y**. If not, we use **N**. When the feature is not mentioned in the article, the column is marked with "–". The column **AAA** identifies if the proposal supports authentication, authorization and accounting. The columns **QoS** and **MOBILITY** consider QoS and client mobility, respectively. Traffic monitoring is classified in column **MONITOR**. We identify if the proposals use the APs exclusively to monitor the network (**EXCL**), or not (**N**). These devices can collect the whole packet as well as wireless medium information (**TOTAL**), or parts of the packet and trivial statistics (**PARCIAL**). The column **FORWARDING** identifies if the proposal controls packet forwarding with a fine (**FINE** – combination of data and headers) or course (**COURSE** – only packet headers) grain. **NETWORK** shows which networks the proposal supports, like 3GPP, LTE, WLAN, and WiMax. The column **MODIFICATION** indicates if the proposals require changes in the AP (**AP**), or in the base station **BS**. Below we analyze these works.

Commercial Wi-Fi systems use centralized controllers to configure and manage the APs, and to optimize network performance [8,19–21]. Hence, there is a separation of the control and data planes using a proprietary southbound interface. These systems claim to support some low-level AP management features such as intelligent client handover, band-steering and mobility support. They also provide proprietary APIs that allow the vendor to get more information about the wireless network behavior. However, such systems most of the time follow a "one solution fits all" approach, in which the provided wireless services tend to be generic. This has lead to a number of proposals in the literature tackling SDN-based solutions for Wi-Fi and wireless networks in general, so that the management of the network can be customized in more details than the existing proprietary solutions.

Our solution does not use "Big AP" or LVAP abstractions, like ODIN [30], SDWLAN [34], "One Big AP" [31], OpenSDWN [33], and OpenRoads [1]. These approaches remove functions (related to Beaconing, Probe and the Association and Authentication process) from the AP, transferring them to the controller, leaving only some basic functions in the AP. This separation allows a program running on the controller to change the behavior of these functions, as shown for example in SDWLAN. However two problems can occur: (1) all the AP's management traffic has to be transmitted over the network to the controller, generating a overload, and (2) the Wi-Fi network timing is in the order of microseconds to a few milliseconds depending on the function, so network latency and controller processing can affect wireless network operation, especially in high traffic or high mobility, and wired network congestion. These problems have not been addressed in these papers. *Ethanol* uses a different approach, leaving the AP with all functions, however the controller is able to register to receive messages triggered by events in the AP and based on them make decisions to change the behavior of the AP. The controller can still request information from the AP and decide actions from this information. *Ethanol* can steer flows through the appropriate network functions and obtain the client's state, which is also a substrate to develop network function virtualization. OpenSDWN extends the virtualization proposed in ODIN to the middleboxes, using SDN as an enabler to the network function virtualization paradigm. This approach is complementary to *Ethanol*, which can benefit from the proposed traffic classification mechanism. *Ethanol* can support all wireless SDN services provided by OpenSDWN. But, only Ethanol supports handling management messages, and interacting with the connected stations. "One Big AP" proposes an abstraction for clients, and focuses on client-transparent handover, because handover is accomplished by moving the attachment of the virtual AP from one physical device to another. This proposal virtualizes the association, and the authentication process, which can generate excessive network traffic. *Ethanol*, on the other hand, focuses on AP management, and uses the default handover mechanism proposed by IEEE 802.11. It, however, allows the administrator to interfere with the handover decision. *Ethanol* can interact with the stations and other network devices.

Ethanol uses however a virtual AP feature that is defined in IEEE 802.11 standard, which consists of creating severalBasic Service Set Identifiers (BSSIDs) in a physical device. In this paper, *VAP*, in *Ethanol* context, refers to this concept.

Similar to Ethanol, there are a number of works on the literature that propose new southbound abstractions for the control of Wi-Fi networks. OpenRF is complementary to *Ethanol*, since OpenRF does not provide control functionalities above the physical layer. SWAN [37] propose a software AP to provide QoS guarantees to users, on the other hand *Ethanol* uses a different abstraction, but can also implement SWAN scenarios: seamless handover and load balancing. *Ethanol* can manage a broader range of Wi-Fi parameters than CoAP [25], Dyson [45] and Ætherflow [28]. For example, Ætherflow deals with the main management events defined in the protocol: probe; authentication; deauthentication; association; reassociation; disassociation; and authorization. *Ethanol* adopts a different approach, i.e., it also handles these events, however, the controller registers in the AP which event it will treat in a publisher–subscriber manner, while unregistered events are treated locally by the AP. In addition, *Ethanol* can handle other management frames like Beacon, Action, and Timing Advertisement. The set of statistics obtained by *Ethanol* also outperforms Ætherflow, since in addition to the metrics used by Ætherflow, *Ethanol* is able to receive the reports of the IEEE standard, defined in Section 4.3.11, as well as device performance metrics such as CPU and memory usage. The *Ethanol* API can change the *hostapd* configuration file, creating non-volatile configuration. Moreover, *Ethanol*'s source code is available on the Internet, and can be adapted (and improved) by the network admin.

Backhauling provides wireless programmability by shifting part of the processing that is performed on the AP to a cloud infrastructure. IETF RFC 5415 proposed CAPWAP[5] that supports two modes of operation: Split and Local MAC. In Split MAC mode, all frames are encapsulated and sent to the controller. In Cisco APs, this is the default mode [46]. In Local MAC, the AP performs the 802.11 Integration function, and the controller does the distribution function. In *Ethanol* both functions are normally done by the AP, the controller registers to be triggered by events. Other works [22,24,47] in the literature adopt similar approaches, executing even MAC functions on the Cloud. Backhauling generates a larger load on the wireless controller and on the network than *Ethanol*, since in *Ethanol* the controller is only a decision point. Hence, Backhauling demands a larger redesign of the wireless APs and the wired network. However, backhauling allows for a higher level of extensibility than *Ethanol*. In *Ethanol* we chose not to perform backhauling in order to reduce the costs of our solution.

CloudIQ [23], SoftCell [35], Mobiflow [29], SoftRAN [36], and OpenRAN [48] are SDN architectures for cellular networks. *Ethanol* can transpose some of these ideas to the Wi-Fi world, but our approach does not transfer all wireless functions to the controller, compromising flexibility for performance.

Today most users roam among networks using different technologies. Proposals that manage heterogeneous networks usually provide only generic control functions, so that they can be supported in every supported wireless standard. EmPOWER [11] and CROWD [26] support LTE and WLAN cells. EmPOWER runs on a Click agent[6] and uses a virtual APs abstraction.

To the best of our knowledge, 5g-EmPOWER is the closest related platform to *Ethanol*. The main differences between *Ethanol* and 5g-EmPOWER are: (1) As *Ethanol* is specialized in WLAN, it presents an already developed and more complete set of primitives, allowing to configure features like mobility domain and security in the AP. In 5g-EmPOWER, mobility is done by reallocating the LVAP to another physical AP; (2) The IEEE 802.11 management messages can be handled by *Ethanol* at the AP or at the controller, while 5g-EmPOWER can handle them only at the controller. Thus, the *Ethanol* approach reduces communication latency and decreases controller overhead. (3) *Ethanol* does not use the concept of LVAP used by 5g-EmPOWER. Instead, the *Ethanol* controller sends tasks directly to the programmed AP. In this way 5g-EmPOWER gives the user the illusion that the user has a private AP, while *Ethanol* shares the SSID among the users; (4) Because LVAP runs on the controller, 5g-EmPOWER can transmit a (theoretically) unlimited number of SSIDs in each physical AP. On the order hand, *Ethanol* relies on the *hostapd* implementation, thus it is limited to broadcast 255 SSIDs, but this limit can be raised by changing a compilation parameter up to the size of the "integer" C type. *Ethanol* controller can use the beacon and messaging features in *Ethanol* to simulate this 5g-EmPOWER feature, but in a less efficient way.

Summarizing, *Ethanol* is a SDN architecture for IEEE 802.11 devices that uses IEEE compliant messages. This allows the architecture to control AP as well as some functions of the stations. Further, since *Ethanol* is focused towards IEEE 802.11, it provides a finer level of control over the devices than its competitors. To the best of our knowledge, *Ethanol* is the first SDN architecture for wireless networks that gathers information from the wireless stations. Some proposal collect client information, for example in [49]. Their proposal is an specialized solution that uses IEEE 802.11k to obtain customer topology information, but this solution is not an SDN architecture such as *Ethanol*. Also it was only tested in a simulation environment (ns3).

7. Conclusions

This paper describes an SDN approach for the control and management of IEEE 802.11 wireless networks, called *Ethanol*, which provides an API for mobility, security, QoS, and virtualization of wireless networks. Besides improved QoS and performance, we argue that SDN-enabled APs will also be used for the creation of context and location

[5] http://www.ietf.org/rfc/rfc5415.txt.

[6] http://www.read.cs.ucla.edu/click/click.

aware services. The main focus of this paper is on the southbound interfaces of *Ethanol*. Broader northbound interfaces, beyond HomeNetRescue [3], will be defined in future work. We present the architecture of a SDN-enabled WLAN, as well as the methods, properties, and events of the control API. We provided a prototype that only depends on the Linux AP implementation, so it can be used on any device that supports that implementation. An analysis has shown that the overhead of *Ethanol* is in the order of 1%–2% in memory and CPU usage, hence it should run on embedded versions of Linux used in some commercial APs.

Ethanol is evaluated on a prototype developed with Linux computer acting as APs. The experiments show that the network performance can be enhanced by programmable APs.

Several use cases, shown in this article and in other articles of the group, show the potential of *Ethanol*. In this article, we propose an algorithm that improves video quality by 45% during handover. Further, *Ethanol* can interact with an application to know when to change the dynamics of the network. In the experiment shown, a CCTV application can obtain 2-3x higher throughput during relevant events. Further, *Ethanol* was used to implement application aware QoS policies, and to detect network failures.

In a previous work [3], we showed that using *Ethanol* can increase the network throughput by up to 131%, reducing wireless transmission delay and jitter by 46% and 24%, respectively. We also showed in [6] that *Ethanol* provides the means to improve the QoE of Web applications, reducing the page load time by at least 25% in the worst case, when compared to non-managed deployments.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior — Brasil (CAPES) - Finance Code 001, CNPq, Brazil (funding agency from the Brazilian federal government), and FAPEMIG, Brazil (Minas Gerais State Funding Agency).

References

[1] K.-K. Yap, R. Sherwood, M. Kobayashi, T.-Y. Huang, M. Chan, N. Handigol, N. McKeown, G. Parulkar, Blueprint for introducing innovation into wireless mobile networks, in: ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures (VISA), 2010, pp. 25–32.

[2] H.D. Moura, G.V.C. Bessa, M.A.M. Vieira, D.F. Macedo, Ethanol: Software defined networking for 802.11 wireless networks, in: IFIP/IEEE International Symposium on Integrated Network Management (IM), 2015, pp. 388–396.

[3] A.R. Alves, H.M. Duarte, J.R.A. Borges, V.F.S. Mota, L.H. Cantelli, D.F. Macedo, M.A.M. Vieira, HomeNetRescue: an SDN service for troubleshooting home networks, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2018.

[4] A.D. Ferguson, A. Guha, J. Place, R. Fonseca, S. Krishnamurthi, Participatory networking, in: Proceedings of the 2Nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, in: HotICE'12, USENIX Association, Berkeley, CA, USA, 2012, p. 2, [Online]. Available: http://dl.acm.org/citation.cfm?id=2228283.2228286.

[5] J.C.T. Guimaraes, H.D. Moura, J.R. Borges, M.A. Vieira, L.F. Vieira, D.F. Macedo, Dynamic bandwidth allocation for home and soho wireless networks, in: 2018 IEEE Symposium on Computers and Communications (ISCC), IEEE, 2018, pp. 00373–00376.

[6] H.D. Moura, D. Fernandes Macedo, M.A.M. Vieira, Automatic quality of experience management for wlan networks using multi-armed bandit, in: 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), 2019, pp. 279–288.

[7] H. Song, Protocol-oblivious forwarding: unleash the power of sdn through a future-proof forwarding plane, in: ACM Workshop on Hot Topics in Software Defined Networking (HotSDN), 2013, pp. 127–132.

[8] I. Cisco Systems, Cisco Wireless LAN Controller Com- mand Reference, Release 7.6, Cisco Systems, Inc, 2013.

[9] Y.-C. Cheng, J. Bellardo, P. Benkö, A.C. Snoeren, G.M. Voelker, S. Savage, Jigsaw: Solving the puzzle of enterprise 802.11 analysis, in: ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM), 2006, pp. 39–50.

[10] P. Machań, J. Wozniak, Proactive handover for ieee 802.11 r networks, in: 2011 4th Joint IFIP Wireless and Mobile Networking Conference (WMNC 2011), IEEE, 2011, pp. 1–7.

[11] R. Riggio, M. Marina, J. Schulz-Zander, S. Kuklinski, T. Rasheed, Programming abstractions for software-defined wireless networks, IEEE Trans. Netw. Serv. Manag. 12 (2) (2015) 146–162.

[12] S. Zehl, A. Zubow, A. Wolisz, BIGAP-A seamless handover scheme for high performance enterprise IEEE 802.11 networks, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2016, pp. 1015–1016.

[13] E. Coronado, J. Villalon, A. Garrido, Wi-balance: SDN-based load-balancing in enterprise WLANs, in: IEEE Conference on Network Softwarization (NetSoft), 2017, pp. 1–2.

[14] L. Sequeira, J.L. de la Cruz, J. Ruiz-Mas, J. Saldana, J. Fernandez-Navajas, J. Almodovar, Building an SDN enterprise WLAN based on virtual APs, IEEE Commun. Lett. 21 (2) (2017) 374–377.

[15] Z. Wang, E.P. Simoncelli, A.C. Bovik, Multiscale structural similarity for image quality assessment, in: Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on, Vol. 2, IEEE, 2003, pp. 1398–1402.

[16] Z. Wang, A.C. Bovik, H.R. Sheikh, E.P. Simoncelli, Image quality assessment: from error visibility to structural similarity, IEEE Trans. Image Process. 13 (4) (2004) 600–612.

[17] J. Epstein, Scalable VoIP Mobility: Integration and Deployment, Newnes, 2009.

[18] Z. Kerravala, The Realities of Wi-Fi Troubleshooting, https://zkresearch.com/research/the-realities-of-wifi-troubleshooting/, 2016, accessed: 29-09-19.

[19] I. Aruba Networks, ArubaOS 6.3.x User's Guide, 0511497-00v6, Aruba Networks, Inc, 2014, p. 1003p.

[20] I. Juniper Networks, Mobility System Software, Command Reference Guide, Juniper Networks, Inc, 2014, p. 723p.

[21] I. Extreme Networks, IdentiFi Wireless User Guide, v9.12.XX, Extreme Networks, Inc, 2014, p. 1016p.

[22] Y. Yiakoumis, M. Bansal, A. Covington, J. van Reijendam, S. Katti, N. McKeown, BeHop: a testbed for dense WiFi networks, ACM SIGMOBILE Mob. Comput. Commun. Rev. 18 (3) (2015) 71–80.

[23] S. Bhaumik, S.P. Chandrabose, M.K. Jataprolu, G. Kumar, A. Muralidhar, P. Polakos, V. Srinivasan, T. Woo, CloudIQ: A framework for processing base stations in a data center, in: International Conference on Mobile Computing and Networking (MobiCom), 2012, pp. 125–136.

[24] P. Dely, J. Vestin, A. Kassler, N. Bayer, H. Einsiedler, C. Peylo, CloudMAC: An openflow based architecture for 802.11 MAC layer processing in the cloud, in: IEEE Globecom Workshops, 2012, pp. 186–191.

[25] A. Patro, S. Banerjee, COAP: A software-defined approach for home WLAN management through an open API, SIGMOBILE Mob. Comput. Commun. Rev. 18 (3) (2015) 32–40.

[26] H. Ali-Ahmad, C. Cicconetti, A. De la Oliva, M. Dräxler, R. Gupta, V. Mancuso, L. Roullet, V. Sciancalepore, CROWD: an SDN approach for DenseNets, in: European Workshop on Software Defined Networks, 2013, pp. 25–31.

[27] R. Riggio, The EmPOWER mobile network operating system, in: ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization, 2016, pp. 87–88.

[28] M. Yan, J. Casey, P. Shome, A. Sprintson, A. Sutton, ÆtherFlow: Principled wireless support in SDN, in: 2015 IEEE 23rd International Conference on Network Protocols (ICNP), 2015.

[29] K. Pentikousis, Y. Wang, W. Hu, Mobileflow: Toward software-defined mobile networks, IEEE Commun. Mag. 51 (7) (2013) 44–53.

[30] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, T. Vazao, Towards programmable enterprise WLANS with Odin, in: ACM Hot Topics in Software Defined Networks, 2012, pp. 115–120.

[31] D. Zhao, M. Zhu, M. Xu, Supporting "One Big AP" illusion in enterprise WLAN: An SDN-based solution, in: International Conference on Wireless Communications and Signal Processing (WCSP), 2014, pp. 1–6.

[32] S. Kumar, D. Cifuentes, S. Gollakota, D. Katabi, Bringing cross-layer MIMO to today's wireless LANs, in: ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, 2013, pp. 387–398.

[33] J. Schulz-Zander, C. Mayer, B. Ciobotaru, S. Schmid, A. Feldmann, OpenSDWN: programmatic control over home and enterprise WiFi, in: ACM SIGCOMM Symposium on Software Defined Networking Research, 2015, pp. 16.

[34] D. Zhao, M. Zhu, M. Xu, SDWLAN: A flexible architecture of enterprise WLAN for client-unaware fast AP handoff, in: International Conference on Computing, Communications and Networking Technologies, 2014, 1–6.

[35] X. Jin, L.E. Li, L. Vanbever, J. Rexford, SoftCell: Scalable and flexible cellular core network architecture, in: ACM Conference on Emerging Networking Experiments and Technologies, 2013, pp. 163–174.

[36] A. Gudipati, D. Perry, L.E. Li, S. Katti, SoftRAN: Software defined radio access network, in: ACM Workshop on Hot Topics on Software Defined Networks, 2013, pp. 25–30.
[37] T. Lei, Z. Lu, X. Wen, X. Zhao, L. Wang, SWAN: An SDN based campus WLAN framework, in: International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace Electronic Systems, 2014, pp. 1–5.
[38] K. Chintalapudi, A. Padmanabha Iyer, V.N. Padmanabhan, Indoor localization without the pain, in: ACM International Conference on Mobile Computing and Networking (MobiCom), 2010, pp. 173–184.
[39] A. Thaljaoui, T. Val, N. Nasri, D. Brulin, BLE localization using RSSI measurements and iRingLA, in: IEEE International Conference on Industrial Technology (ICIT), 2015, pp. 2178–2183.
[40] G. Mao, B. Fidan, B.D. Anderson, Wireless sensor network localization techniques, Comput. Netw. 51 (10) (2007) 2529–2553.
[41] G. McGrath, P.R. Brenner, Serverless computing: Design, implementation, and performance, in: 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), IEEE, 2017, pp. 405–410.
[42] C. Negus, Docker Containers, second ed., Addison-Wesley Professional, 2015.
[43] S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, A.C. Arpaci-Dusseau, R.H. Arpaci-Dusseau, Serverless computation with openlambda, in: 8th *USENIX* Workshop on Hot Topics in Cloud Computing (HotCloud 16), 2016.
[44] D.F. Macedo, D. Guedes, L.F.M. Vieira, M.A.M. Vieira, M. Nogueira, Programmable networks – from software-defined radio to software-defined networking, IEEE Commun. Surv. Tutor. 17 (2) (2015) 1102–1125.
[45] R. Murty, J. Padhye, A. Wolman, M. Welsh, Dyson: An architecture for extensible wireless LANs, in: USENIX Annual Technical Conference, 2010.
[46] I. Cisco Systems, Flexconnect, Cisco Systems, Inc, 2017. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/wireless/controller/8-1/Enterprise-Mobility-8-1-Design-Guide/Enterprise_Mobility_8-1_Deployment_Guide.html.
[47] A. Kenny, Q.-D. Ho, T. Le-Ngoc, eWV: An evolvable platform for versatile control in software-defined wireless networks, in: IEEE International Conference on Communications Workshops, 2016, pp. 724–729.
[48] M. Yang, Y. Li, D. Jin, L. Su, S. Ma, L. Zeng, Openran: a software-defined ran architecture via virtualization, ACM SIGCOMM Comput. Commun. Rev. 43 (4) (2013) 549–550.
[49] P.A. Frangoudis, G.C. Polyzos, Reputation-based crowdsourced wi-fi topology discovery, Comput. Netw. 79 (2015) 1–16.