# Ethanol: Software Defined Networking for 802.11 Wireless Networks

**Henrique Moura***, **Gabriel V. C. Bessa***, **Marcos A. M. Vieira***, **Daniel F. Macedo***
*Computer Science Department
Universidade Federal de Minas Gerais, Brazil
Emails: {henriquemoura, gabrielvcbessa, mmvieira, damacedo}@dcc.ufmg.br

***Abstract*—Wireless Networks have become ubiquitous and dense to support the growing demand from mobile users. To improve the performance of these networks, new approaches are required, such as context and service aware control algorithms, which are not possible on today's closed proprietary WLAN controllers. In this work, we propose Ethanol, a software-defined networking architecture for 802.11 dense WLANs. This paper describes the benefits of programmable APs, and proposes Ethanol, an architecture for network-wide control of QoS, user mobility, AP virtualization, and security on 802.11 APs. The proposal is evaluated on a prototype using off-the-shelf APs over three use cases.**

## I. Introduction

Wireless Networks have become ubiquitous and dense. In a recent study [1], more than 90 percent of U.S. colleges intend to increase their wireless infra-structure to support the demand of mobile applications. The same trend is expected in home networks. With the Internet of Things, more devices will connect to the network. Cisco predicts that 50 billion things will connect to the Internet by 2020 [2]. At the same time, emerging high-speed network standards (e.g. 802.11ad) are migrating to higher frequencies, which are absorbed by walls, thus requiring the deployment of multiple access points. Thus, future networks must be optimized to increase their scalability and effectiveness in dense environments through a more refined control of all the devices in the network.

Current management architectures for WLANs employ proprietary controllers, which perform network-wide optimizations such as the selection of best operational channels, the adjustment of transmission power at each AP, faster client mobility and enhanced traceability, as well as security and QoS policy enforcement. However, those controllers only manage compatible devices, usually of a single manufacturer, since they rely on proprietary interfaces and proprietary MIBs. Further, current wireless network controllers for WLAN remain closed and (mostly) proprietary [3], and the industry has little incentive to change.

Software Defined Networking (SDN) is a recent network paradigm that restructures computer networks by separating the control plane from the data plane, bringing dynamic programmability to the networks [4]. SDN permits network elements to run simple programs, and is said to reduce the complexity of network configuration and network management while providing leverage to innovation in a simpler way. SDN advocates the possibility of: $(i)$ more evolvable networks, as we should be able to implement new features without changing the platform; and $(ii)$ greater network efficiency, since more advanced (context and application aware) algorithms can be easily deployed (usually in run-time).

However, in existing SDNs (e.g. OpenFlow), the data plane is programmable only on switching elements. We advocate that SDN-enabled access points will be able to improve the management of WLANs even further than proprietary controllers. Having an open API will allow the deployment of context and application aware control algorithms, an impossible task on current wireless controllers. Further, an open API for access points will allow rapid innovation, accelerate the deployment of new network services and foster the competition among vendors. In the same way that SDN has generated many innovative applications on the wired domain, we believe that SDN-enabled access points will accelerate the innovation on WLAN management and operation.

This paper proposes an SDN architecture for dense IEEE 802.11 Wireless LANs called Ethanol. Ethanol refactors the control plane functionalities between the APs and the controller. The centralized controller actuates on APs, controlling features such as quality of service, client mobility and association/disassociation, link-level parameters and current state, and virtual APs. The proposed architecture was prototyped on commodity home routers, and the performance of Ethanol was evaluated on three case studies involving quality of service, the control of client association, and ARP packet suppression.

The remaining of the text is organized as follows. Sections II and III provide a background on SDN and the challenges of SDN on wireless networks, respectively. The Ethanol architecture and its components are described in Section IV, followed by its implementation in Section V. The implementation and the results obtained on a testbed are presented in Section VI. The related work is discussed in Section VII. Finally, Section VIII presents our conclusions and future work.

## II. Background on SDN

In this section, we provide an overview of the existing control methods for SDNs and for wireless networks.

SDNs are characterized by the existence of a control system (software) that controls the switching fabric of routers and

switches (the data plane hardware) through a well defined programming interface. One programming interface that adheres to the SDN philosophy is OpenFlow [4], which is tailored to control routers and switches. OpenFlow is currently the *de facto* standard in SDN programmability.

The forwarding operation of OpenFlow-enabled devices follows a simple principle: each packet received on one of the switch's interfaces is inspected and generates a query to the forwarding table. If the query does not succeed, the packet can be dropped or forwarded to the controller, which decides how to handle it. Once the destination of the packet is identified, it is switched to the destination port, where it is queued for transmission. Thus, in OpenFlow-based SDN, the programmable operations are *forward*, *drop*, *modify header* and *send to controller*, the latter required to deal with unknown flows. Besides actions, the architecture provides per-flow counters maintained for each flow table, flow entry, port, queue, group, group bucket, meter and meter band.

The OpenFlow controller communicates with OpenFlow switches over a secure channel. The protocol instructs OpenFlow-compatible switches to update their forwarding table to take different actions on various network flows. Thus, the controller contains all the logic on how the forwarding table is updated, and the network element merely executes the forwarding rules generated by the controller.

SDN advocates the use of simple but software-programmable network devices, so the operator can develop its own software, and get it deployed at much shorter cycles than today's experience with proprietary state-of-the-art hardware-centric network devices.

The SDN paradigm and Openflow were proposed and engineered aiming infrastructure networks, more specifically wired networks. The adaptation of these concepts to the wireless context is not straightforward, as described in the next section.

## III. Current challenges for software defined wireless networks

Wireless networks have their particularities with regards to wired networks, mostly related to the highly variable characteristics of the links and the mobility of the users. Thus, the abstractions and primitives proposed by wired SDN approaches (e.g. OpenFlow) are not sufficient to perform the proper control of a wireless network. This section presents the challenges of wireless networks and how they influence the design of a programmable control plane.

### A. Variable link characteristics

Unlike most wired networks, where links have fixed bandwidth and error rates, in wireless networks those characteristics may vary for every packet transmission. Thus an SDN controller should be aware of link characteristics (e.g. bitrate, delay, loss rate, etc) for tasks such as routing in mesh networks, channel allocation, or scheduling of transmissions of the terminals.

Further, transmission quality is greatly affected by congestion and interferences, so identifying the local topology surrounding a client node is important. JIGSAW [5] highlighted the difficulties on achieving a global view on a wireless network, such as the correct positioning of monitors, large scale synchronization, frame unification and multi-layer reconstruction. The IEEE 802.11k (Radio Resource Measurement of Wireless LANs) standard provides mechanisms for access points and stations to dynamically measure (and report) available radio resources. In an 802.11k enabled network, both clients and access points can send neighbor reports, beacon reports, and link measurement reports to each other, allowing them to have a better understanding of the wireless medium.

### B. Node mobility

Software defined wireless networks (SDWN) should be able to manage node mobility, controlling which users should associate to a certain access point, and identifying when a handoff to another AP is about to take place. This is desirable in order to implement load balancing and topology control, as well as to reduce the handoff time by speeding up management tasks such as authentication and address assignment.

Node mobility is a significant issue in wireless networks, as shown by the many standards addressing it. IEEE Inter-Access Point Protocol standard (802.11f) provides wireless access point communications among multivendor systems designed for the enforcement of unique association throughout an Extended Service Set and for secure exchange of station's security context between the current and the new access point (AP) during the handoff period. Similarly, IEEE 802.21 deals with handovers between heterogeneous networks (e.g. WiMax, 802.11, Bluetooth). Also, IEEE 802.11r supports fast and secure handoffs from one access point to another, performing Fast BSS Transitions with security key negotiation.

### C. Quality of service

The current Openflow specification has very basic QoS support. OpenFlow mainly provides means to set a flow to a queue. OpenFlow allows the use of per-flow meters to implement various simple QoS operations, such as rate-limiting, but this is an "optional" feature. The OpenFlow protocol is able to manage VLAN tags and the IP priority bits, however nothing is done at the link level.

Those configurations alone do not ensure a minimum quality of experience (QoE) to the user. In a WLAN, QoS is determined by roughly three factors: (1) the service architecture (how end-to-end service is provided to the user), (2) the core network performance, and (3) the service provided at the "wireless part", which is the combination of the wireless link and the capabilities of the terminals (wireless clients).

In WLANs, IEEE 802.11e enhances the 802.11 Media Access Control (MAC) layer with service differentiation ([6, 7]), and adds error-correcting mechanisms for delay sensitive applications (e.g. voice and video). But 802.11e alone only handles QoS parameters in a BSS. Using an SDWN, a controller should be able to configure the QoS parameters of the wired and wireless links. Again, ensuring a proper QoS in wireless networks is more complex than in wired networks, since the

packet delivery time and the throughput can vary due to the dynamic nature of the links, as well as the contention of the wireless link, since it is a broadcast medium.

### D. Virtualization

Network virtualization allows multiple isolated logical networks to share the same physical infrastructure even if they use different addressing and forwarding mechanisms. FlowVisor uses SDN to achieve this segmentation, slicing five dimensions [8] of a switching element: bandwidth, topology, traffic, device CPU and forwarding tables. The same dimensions, with the exception of the forwarding tables, are present in APs, and could be subject to control by a centralized entity.

The use of virtualization allows the coexistence of a research wireless networks and production networks at the same physical location, providing not only isolated applications but also management. Also we can benefit of faster provisioning, enabling elastic capacity to provide system provisioning and deployment at a moment's notice by the activation of cloned virtual access point. But there are drawbacks too, as a wireless device still has a limited number of physical radios.

### E. Security

Ethane [9], the ancestor of OpenFlow, was created to improve user access control. OpenFlow, however, does not emphasize security. In a wireless environment, security is an important topic because anyone in range can eavesdrop or disrupt the wireless network.

An SDN environment provides monitoring capacity to the network administrator, which allows a clear vision of the network status, supplying means to detect intruders and abnormal activities. Online traffic can be compared to previous traffic or to stored good/malicious traffic patterns and statistics, and the controller can decide if this traffic matches expected behavior. OpenSketch [10] provides a simple three-stage pipeline (hashing, filtering, and counting) that could be used to provide more refined flow statistics.

One important security task on an enterprise network is the detection of rogue (unauthorized) APs, since those may degrade the performance of official APs and expose the network to unwanted access. The detection of rogue APs requires cooperation among the company's access points, which is facilitated by the centralized control provided by SDWN. Further, the reaction to rogue APs shows the interdependence of wired and wireless SDN, since the only way to avert those APs is to deny packet forwarding coming from them.

### F. User Location

The location of the user is an important element for location-aware services. [11] show that indoor localization is possible with no pre-deployment effort using the EZ localization algorithm. This algorithm requires as input that stations perform RSS (Received Signal Strength) measurements of known APs or a location fix (e.g. a GPS lock near doors or windows). RSS readings can be requested to the stations using 802.11k.

Localization is also important for handoff decisions and network security. A client can provide the controller an indication of its movement direction, allowing handoffs to be more precise. While rogue access point and attacker detection algorithms identify unwanted access and perform virtual countermeasures, localization information could be used to simplify the identification of the intruders, which is essential to perform physical countermeasures.

## IV. Ethanol Architecture

Ethanol is a SDN-based architecture for dense IEEE 802.11 WLANs, for example a network with many APs and clients (a campus, an enterprise network, a network of things in a home). Ethanol allows the development of custom control software, allowing operators to run services that fit their needs. Besides forwarding, the controller can also control node mobility, authentication, virtual networking, QoS, and even user localization (as explained in the previous section). Ethanol adopts the following design goals: (*i*) supports IEEE 802.11 as well as Ethernet NICs; (*ii*) does not require changes on the terminals (data collected from terminals relies on 802.11 standards); (*iii*) provides APIs for node mobility, AP virtualization, WLAN security, and QoS (on WiFi and Ethernet).

The Ethanol architecture is composed of two types of devices: the controller and Ethanol-enabled APs. The controller runs on a computer in the wired network or virtualized in the cloud. The controller can also control OpenFlow-enabled switches if necessary. The Ethanol APs are commodity wireless routers that are modified to run Ethanol code. Ethanol is very slim, so it can be deployed on cheap home APs.

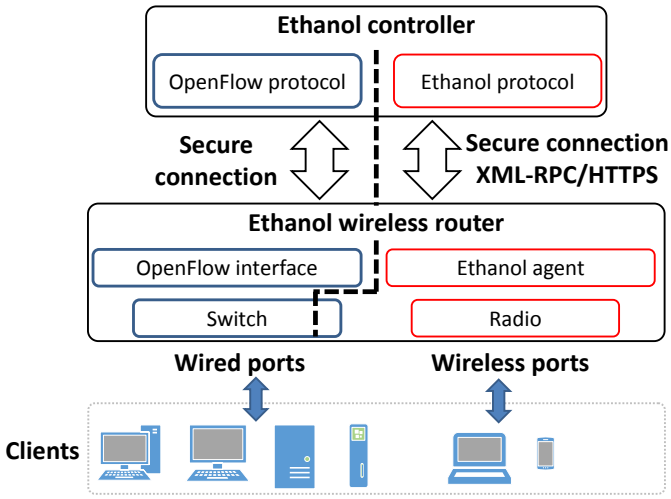


Fig. 1. Ethanol Access Point Implementation

Figure 1 depicts the Ethanol AP architecture. An Ethanol AP has two parts: wired and wireless components. A wired component is a configurable switching element that supports the OpenFlow protocol. Since OpenFlow does not provide a control interface for QoS and for wireless components, the Ethanol Agent adds this functionality to the AP, using an

additional API for controlling wireless links and for defining the QoS of wired flows. This agent receives commands from the Ethanol Controller via a secure channel.

### A. 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. For example, an Ethanol AP and a flow are examples of a physical and a virtual entity, respectively. Those entities have observable and/or configurable *properties*, such as the ESSID, or the number of clients associated in an AP. The properties are accessed by the controller using *get*/*set* methods. Finally, entities can have *events*, which generate calls to the controller so that it can respond appropriately. One such event could be a wireless client requesting an association. Figure 2 shows the entities, properties and events of Ethanol. To improve readability we have omitted all getter and setter methods. 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.

Some of the proposed properties, methods, and entities may not be feasible on some production APs, however we chose to specify the architecture without taking into account the limitations of existing equipment. Future hardware could be developed with this specification in mind, in a trend similar to what happened with SDN: its first specifications were limited to functions implementable in existing hardware, and now vendors are proposing chips tailored for SDN operations [12]. Next we provide a brief description of these entities.

*1) AccessPoint entity:* This entity represents physical devices. An AccessPoint can have one or more physical radios, represented by the Link entity, and one or more virtual access points running on the AP (VirtualAccessPoint class). The AccessPoint entity has three main attributes: $beaconInterval$ (affects the frequency of the beacons), $fastBSSTransition\_compatible$ (if the access point is compatible with 802.11r) and $802.11b\_preamble$ (if the preamble is long or short).

To support QoS, we can query the state of the queues, as well as change the queues' priority and queuing disciplines (schedulers). The methods available in this entity allow the creation and destruction of virtual access points, as well as to determine the state of the environment: return the modes supported by the NIC (e.g. ad hoc, infrastructure), and request an interference map, for example to pinpoint rogue APs.

*2) Link entity:* This entity configures the physical interface of the AP. It has attributes such as channel, available bit rates, transmitter power output, and power saving mode. It also gathers link statistics and other information of the NIC.

*3) VirtualAcesssPoint entity:* A physical device can have zero or more virtual access points. If it is zero, the device is not providing any service. We can also configure a virtual AP but keep it disabled for future use or fast startup. Stations connect to a virtual AP, and a group of virtual access points forms a Network. If $broadcastSSID$ is disabled, then such virtual AP does not broadcast its SSID. Also this entity 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 802.11e (e.g. maximum and minimum contention window values, arbitration inter-frame spacing – AIFS) and admission control parameters. Each virtual AP also has its own security parameters.

User association and authentication generate events to 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 802.11r) or for probe requests.

*4) Network entity:* This entity represents a network and its $SSID$. The network may contain several VirtualAccessPoints. It provides methods for the association and dissociation of APs with the network, as well as a method to request a user handoff with 802.11r.

*5) Station entity:* This entity represents a user connection (station). All data is collected using messages from existing 802.11 standards. The entity contains information such as the MAC and IP addresses, and if the station is 802.11e capable. The AP 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, packet loss, delay and jitter.

The entity also provides measurements generated by clients supporting 802.11k. 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, failed/retry counts, etc) using $getStatistics$ method, among others.

*6) Flow entity:* This entity originated from the OpenFlow specification. It has the OpenFlow methods, counters and events. It was expanded with enhanced flow monitoring for wireless traffic to report the delay, loss, and jitter of the flow. Those modifications are highlighted in red in Figure 2.

## V. Ethanol Implementation

An Ethanol prototype was implemented in order to evaluate the basic functions of the architecture. Due to time restrictions and limitations of the hardware and operating system of the APs used in the prototype, we implemented a subset of events and methods described previously. As future work we will implement more functions of Ethanol and employ more capable APs, in order to test a larger part of the architecture.

The Ethanol controller is a Linux computer using POX, which was modified to also handle the Ethanol messages encoded with XML-RPC over HTTPS. The controller runs services presented in the use cases shown in the next section. The Ethanol-enabled APs were implemented on cheap

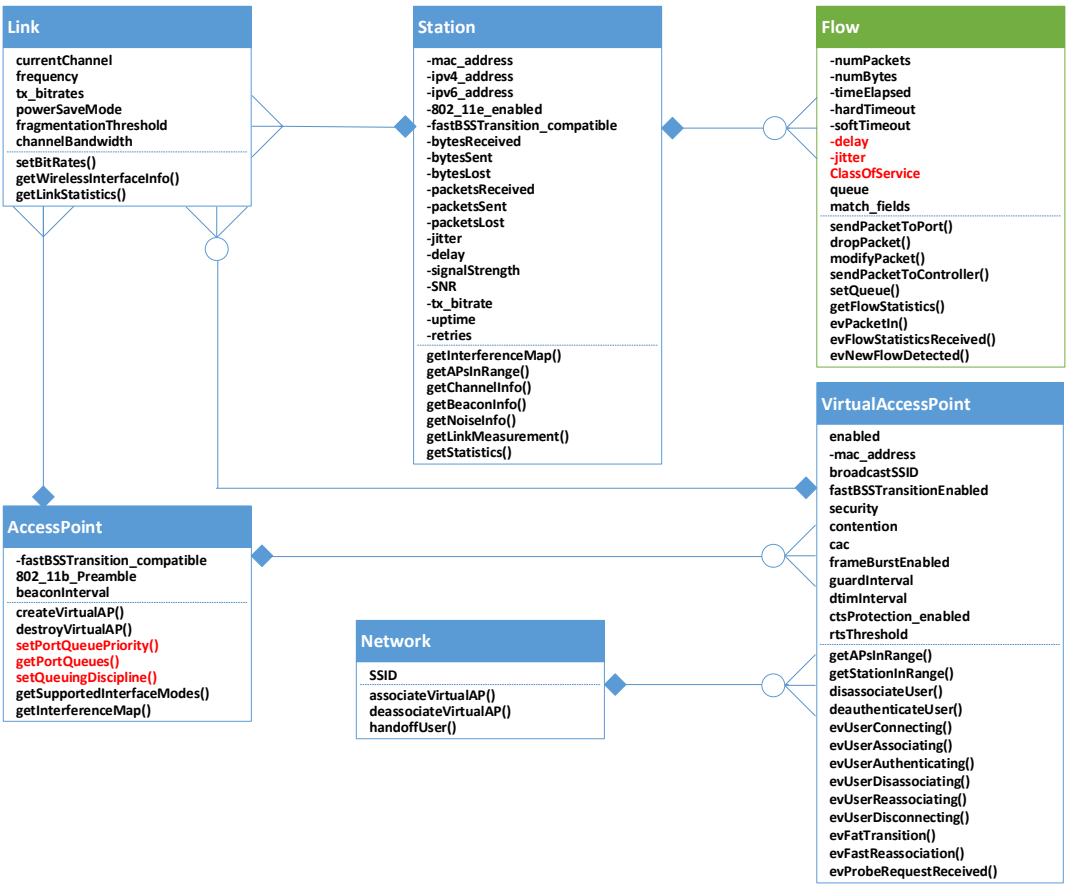


Fig. 2. Ethanol control API model. Green entity indicates current OpenFlow model. Elements in red indicate extensions to OpenFlow.

commodity wireless access points, connected to the controller using ethernet. The chosen model was a Broadcom WRT54GL router running OpenWRT, a linux distribution for embedded devices. Its hardware provides one 802.11b/g Wi-Fi interface and a 4+1 100Mbps Ethernet switch. We installed Pantou, a userspace OpenFlow router that supports version 1.0 of the specification.

Ethanol exploits the QoS elements of Pantou and Linux. Pantou uses Hierarchical Token Bucket (HTB) implementation as its main QoS mechanism. HTB is a packet scheduler that supports priority queuing as well as maximum and minimum rate allocations. Ethanol can control outbound traffic on a link using HTB by associating each flow to a queue. We also modified the Linux wireless subsystem in order to control client association. In Ethanol, the AP first sends a message to the controller asking whether a client should be allowed to associate.

## VI. Experiments

This section describes the experiments performed on the prototype.

### A. Load-Aware Client Association

One way to improve the throughput of dense wireless networks is to control which client is associated to each AP. Suppose that a dense network has two access points, both covering the same location, in order to accommodate a larger number of clients. Those APs are allocated to non-interfering channels, and the objective is to assign the same amount of clients to both APs for load-balancing. Nowadays, clients associate based on signal strength. So, if an AP has a slightly better signal, most clients will attempt to connect to the AP with the strongest signal, generating a load imbalance. Commercial wireless controllers use proprietary algorithms that rely on generic network behavior to control the association process. The network administrator can only set a few parameters, but cannot define the behavior he/she really wants.

In Ethanol, the controller is notified whenever new association attempts occur, as shown in Algorithm 1. So, the controller is able to deny the association or redirect it to other APs in order to perform load balancing (using 802.11r), or using application-specific rules. In this experiment we implement a load-balancing scheme where the controller divides the number of clients equally among APs using only the disassociation feature. Each association request is sent to the controller, which decides if the client should be allowed to associate to the AP based on the actual number of stations already associated (the fewer the better). Observe that Ethanol does not need to modify the client's software to control

the distribution of clients in each AP. On this experiment, our implementation does not attempt to provide a smooth transition as in 802.11r. Besides the number of clients, a controller could steer clients to/from specific APs based on other metrics, such as their link quality, their communication speed (e.g. 6Mbps or 54Mbps), or the type and amount of traffic of each client.

**Algorithm 1** Load aware association control

```
1: function evUserConnecting(userMAC, ap)
2:     apsInRange ←
3:     Station.getStationByMAC(userMAC).getAPsInRange()
4:     minClients ← argmin_{ap ∈ apsInRange} {ap.numClients()}
5:     if ap.numClients() = minClients then
6:         return true                          ▷ Allow connection
7:     else
8:         return false      ▷ Chosen AP has more clients than minimum
9:     end if
10: end function
```

In this experiment, when a wireless client $C$ wants to connect to an AP, first it has to go through the authentication process: $C$ sends an Authentication Request to the AP, which responds with an Authentication Reply. Afterwards, $C$ sends an Association Request. The AP processes the Association Request and sends an Association Reply granting association. We can control the client's distribution in the APs by aborting the association process, forcing the $C$ to look for other APs. Refusing association or re-association requests and forcing client disconnection can make the overall handover slower, but there is no other way to control the association without changes in the clients. Fast BSS transition, as proposed by IEEE 802.11r, relies on a client decision to change access points.

The experimental setup is composed of two Ethanol APs covering the same area, providing the same ESSID. Figure 3 shows the number of associated clients on two Ethanol APs, where a new wireless client attempts to associate every three seconds. The graph shows a monotonically increasing step curve for both APs, where the number of connected clients differs by at most two clients. The discrepancies occur because our implementation allows non-blocking reads at the controller in order to reduce the response time for the client. As a consequence, two requests arriving at the same time might select the same access point.

### B. Quality of Service

Quality of Service (QoS) is an important requirement for dense networks. Suppose there are two clients communicating with the AP. The first client wants to watch a video, while the second wants to participate in a teleconference. These two types of traffic require certain bandwidth and latency requirements to be met in order to provide an acceptable user experience. Thus, the access points should have means to prioritize those flows.

The OpenFlow specification provides limited support for QoS. Queue configuration is still optional in OpenFlow 1.4, thus being vendor dependent. Nowadays, OpenFlow allows to map flows into queues that must be configured using an external tool. On existing access points using OpenWRT, it is possible to map a flow to a queue, however this queue must be manually configured. [13] proposed a modification on Pantou allowing a modified OpenFlow to control linux queues. The Ethanol agent provides an interface to configure these parameters also, thus allowing the controller to provide per-flow programmability of QoS parameters.

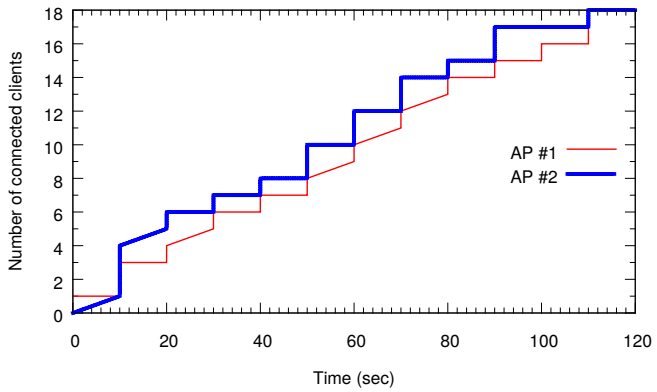


Fig. 3. Controlling client association in a 802.11 network

We evaluate the QoS capabilities of Ethanol in this scenario. This is an important feature for enterprise networks, which require flow prioritization when there are network intensive tasks running concurrently with more delay-driven services. For example, a VM migration among servers should have lower priority than the wireless clients accessing the Intranet.

**Algorithm 2** Associating flows to a queue

```
1: function evPacketIn(event)
2:     cos ← getCoS(event.flow.IP_source)        ▷ Identify Class of Service
3:     queue ← matchCoS(cos)
4:     event.flow.setQueue(queue)          ▷ Map this flow to the correct queue
5: end function
```

For this experiment, an Ethanol AP is connected to two wired clients and a server, and one client is associated on the wireless link. This configuration emulates an enterprise network, where wireless and wired clients want to access an Intranet resource. We ensure flow prioritization using HTB scheduling, so that the throughput of the flows should be proportional to their assigned allocations.

We set up three queues at the Ethanol switch, with rates $Q_1 = 6Mbps$, $Q_2 = 3Mbps$ and $Q_3 = 1Mbps$. When a new flow is detected (PacketIn events), we assign it to the proper queue based on its source address, as shown in Algorithm 2. All clients connect to the server using TCP.

At first, one Ethernet client $E_1$ and one wireless client $W_1$ are active, and associated to queues $Q_1$ and $Q_2$ respectively. As expected, $E_1$ received $\frac{3}{9}$ of the capacity and $W_1$ received $\frac{6}{9}$ of the capacity. At 120 seconds, $E_2$, which is associated to $Q_3$, starts transmitting. The bandwidth allocations are now $\frac{6}{10}$, $\frac{3}{10}$ and $\frac{1}{10}$ of the link capacity, as expected. Thus, Ethanol provides per-flow QoS programmability. Figure 4 shows the throughput over time.

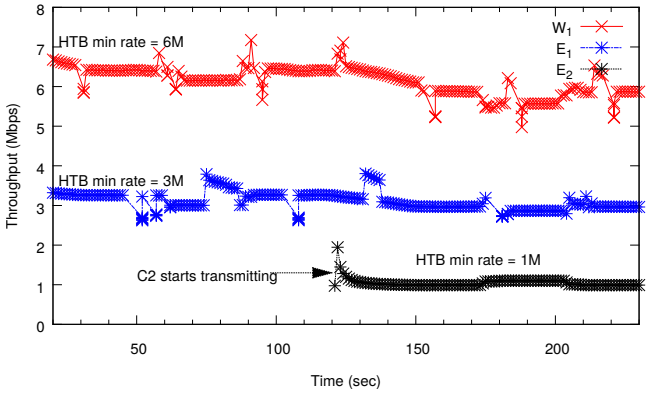


Fig. 4. Controlling traffic to a server

**Algorithm 3** Diminishing ARP Overhead

```
1: function EVPACKETIN(event)                          ▷ event is an OpenFlow parameter
2:     packet ← event.parsed
3:     mac_dst ← packet.mac_dst
4:     IP_dst ← packet.IP_dst
5:     mac_src ← packet.mac_src
6:     IP_src ← packet.IP_src
7:     macToPort[mac_src] ← event.port                 ▷ maintains macToPort mapping
8:     arpTable[mac_src] ← packet.IP_src               ▷ updates ARP table
9:     if packet.type! = ARP then                      ▷ all non ARP packets must be transmitted
10:        (new Flow(event)).apply()
11:    else                                            ▷ treat every ARP packet
12:        if !findIP(arpTable, IP_dst) then
13:            flood(packet)                           ▷ don't know destination, then flood
14:        else                                        ▷ if destination is known, then:
15:            if packet.arp = REQUEST then            ▷ (a) respond to the sender
16:                SendArpReply(mac_dst, mac_src, macToPort[mac_src])
17:            else                                    ▷ (b) transmit response directly to the destination
18:                SendArpReply(mac_src, mac_dst, macToPort[mac_dst])
19:            end if
20:        end if
21:    end if
22: end function
```

### C. ARP Overhead

Cheng [5] et. at. analyzed traces of a WiFi campus network. They concluded that ARP (Address Resolution Protocol) packets consume almost 10% of the *air time* of wireless links. ARP traffic is sent in broadcast, and all ARP broadcasts from the wired network are also sent in broadcast on the wireless channel. Moreover, this traffic scales with the user population, while the wireless capacity remains constant.

Since the controller has knowledge of the IP and MAC mapping, the controller can solve the ARP requests. Thus, by filtering ARP traffic on the APs, it is possible to free up to 10% of the air time.

In this experiment we avoid unnecessary ARP broadcasts by controlling the transmission of ARP messages using OpenFlow. The benefit of Ethanol here is the fact that wired and wireless ports of the AP are controlled using OpenFlow.

Commercial wireless controllers provide solutions based on proxy ARP that partially address the problem shown in [5]. Cisco's controller, for instance, can act as a proxy ARP to the network requests, generating ARP replies, however the replies are broadcast to the entire network, and this approach does not work with static IP addresses. Aruba's controller has a setup called "virtual AP" that does not forward broadcast traffic to the controller, in such case ARP packets are not handled.

On this experiment the Ethanol controller captures all ARP packets and performs simple matches on its internal ARP table. This behaviour is captured by Algorithm 3. If the controller does not recognize the IP address, it floods the request to all ports (line 13). If the controller already knows the IP address (line 16) or when it receives the ARP Reply packet (line 18), the controller generates a unicast message to the requesting node. This method reduces the number of packets that are transmitted in broadcast and also filters ARP reply messages. All other traffic is transmitted as they are (line 10).

To show this behavior we set up a simple experiment: we connected a wireless client running a packet analyzer (tcpdump) to the Ethanol AP. This client captures all incoming and outgoing packets on its wireless interface. We have ethernet clients generating traffic to other wired clients and also to our wireless client. We configured the controller to switch the ARP filter algorithm on and off.

Figure 5 shows the total amount of traffic (inbound and outbound) as well as ARP traffic on the wireless link. When the controller is not filtering ARP packets, the wireless client receives those packets on its wireless interface. Most of those packets are wasting bandwidth, except for the few requests directed to the wireless client. On the first 120 seconds, we see ARP traffic on the wireless side. When the controller starts to manage the ARP Requests (gray area on the figure), the ARP traffic goes almost to zero, because the controller knows the wireless client's MAC address. The actual traffic to and from the wireless client keeps the table up to date. The only ARP traffic seen on the wireless medium when the ARP limiting algorithm is active is requests from the wireless client, and the first request to this client.

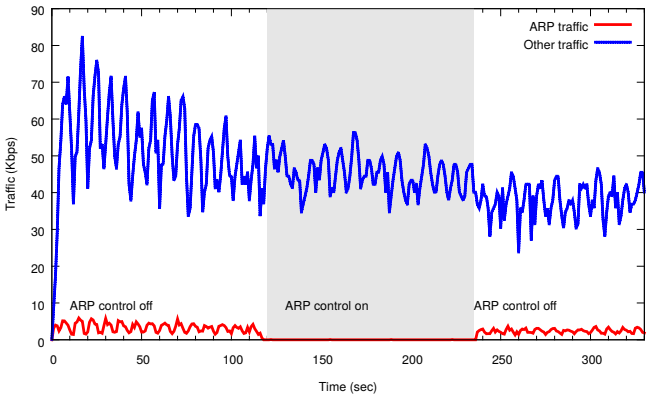


Fig. 5. Controlling ARP overhead on the wireless link

## VII. Related Work

The Ethane [9] project presented an enterprise network architecture which enforces a single fine-grained network policy using a centralized domain controller. This domain controller would compute the flow tables to be installed in each

of the enterprise switches. Ethane is considered the precursor of OpenFlow. Our work is similar to Ethane since it controls the access of clients, however while Ethane was designed for wired networks, while Ethanol focuses on wireless networks.

In OpenRF [14], the APs cooperate to reduce the inter-flow interference using the precoding vectors in MIMO. The controller sets the precoding vectors that reduce interference among APs, while the APs schedule the transmissions of their clients in order to minimize inter-AP interference. OpenRF displays some of the innovative control that can be performed with SDN on the wireless links. It is complementary to Ethanol, since OpenRF does not provide control functionalities above the physical layer.

OpenRoads [3] is a mobile wireless network platform that enables the experimental research and realistic deployment of networks and services using virtualization. OpenRoads proposed OpenFlowWireless, which is built on top of OpenFlow. The main extensions are the ability to slice the datapath using FlowVisor [15] and SNMP. While their focus is on virtualization, Ethanol also addresses QoS, security and mobility.

ODIN [16] employs SDN concepts to tackle client mobility in WLANs. ODIN creates one virtual access point by client, thus it is possible to transparently migrate clients among APs and perform load balancing based on client migration. However, operations such as traffic shaping and QoS enforcement are not possible in ODIN.

IETF proposed CAPWAP, a protocol for the control of APs using a centralized controller [17]. CAPWAP, however, shifts the policy enforcement actions into the controller, as every received frame is tunneled and sent for processing at the controller. Technology-specific management messages are supported, though not defined in the standard. In Ethanol, on the other hand, the controller is a decision point only, and the APs enforce the policies defined by the controller.

APs in a CloudMAC [18] architecture just forward MAC frames between virtual APs and IEEE 802.11 stations and their network functions are placed on the cloud. Our work is orthogonal to CloudMAC. Meanwhile, Ethanol keeps some intelligence in the device, thus not all MAC frames need to traverse the network. As CloudMAC, our approach allows control over configuration commands and a smooth transition from a traditional WLAN, as our design supports all standard WLAN management tools.

CloudIQ [19], SoftCell [20], Mobiflow [21] and SoftRAN [22] are SDN architectures for cellular networks. CloudIQ centralizes all data and control plane processing on a single controller. SoftCell provides high-level fine-grained service polices throughout cellular core networks, even aggregating traffic along multiple dimensions. SoftRAN, on the other hand, puts some functionalities on the individual base stations. For example, frequently varying parameters are performed at the radio element since it has a more up-to-date view of the local state. SoftRAN architecture ensures that the delay and backhaul bandwidth between the controller and the radio element does not negatively impact performance. MobiFlow proposes a control API called *MobileFlow stratum*, similar in purpose to OpenFlow, which interoperates with legacy cellular deployments.

OpenVirteX [23] extends FlowVisor and provides an Infrastructure as a Service (IaaS) for SDNs, enabling operators to create and manage virtual SDNs. Such idea can be transposed to manage virtual access points in a SDWN, so they can be treated as services that can be instantiated, migrated, and deleted. Vitro [24] proposes an integrated architecture for enabling virtualization in wireless sensor networks, decoupling the applications running on physical nodes from the physical sensor deployment. This virtual sensor networking allows dynamic cooperation among nodes. This idea can be transposed to a 802.11 wireless networks using Ethanol.

## VIII. Conclusions

This paper described the architecture and a prototype of Ethanol, an SDN approach for the management and control of dense wireless networks. Ethanol extends the SDN concept to allow the programmability of wireless APs, by providing an API for QoS, security, mobility and virtualization of wireless networks. Besides improved QoS, performance and security, we argue that SDN-enabled APs will also be used for the creation of context and location aware services. We present the architecture of a SDN-enabled dense WLAN, as well as the methods, properties and events of the control API.

Ethanol was evaluated on a prototype developed with cheap, off-the-shelf APs. The experiments indicated that the network performance can be enhanced by programmable APs, allowing an even distribution of clients among APs, the filtering of unwanted traffic, as well as the implementation of QoS policies specific to the wireless medium.

The main focus of this paper was on the southbound interfaces of Ethanol. Its northbound interfaces will be defined in future work. We also plan to implement a larger subset of the Ethanol API, and to evaluate those features on larger networks with more APs and clients.

## Acknowledgements

The authors would like to acknowledge CAPES, CNPq and Fapemig, funding agencies from the Brazilian federal and state government, for financing this research.

## References

[1] "Campus computing national survey of information," http://www.campuscomputing.net, 2013.

[2] D. Evans, "The internet of things – how the next evolution of the internet is changing everything," http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf, 2011.

[3] K.-K. Yap, R. Sherwood, M. Kobayashi, T.-Y. Huang, M. Chan, N. Handigol, N. McKeown, and 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.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[5] Y.-C. Cheng, J. Bellardo, P. Benkö, A. C. Snoeren, G. M. Voelker, and 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.

[6] S. Choi, J. del Prado, N. Sai Shankar, and S. Mangold, "IEEE 802.11e contention-based channel access (EDCF) performance evaluation," in *IEEE International Conference on Communications (ICC)*, 2003, pp. 1151–1156 vol.2.

[7] S. Mangold, Z. Zhong, G. R. Hiertz, and B. Walke, "Ieee 802.11e/802.11k wireless lan: Spectrum awareness for distributed resource sharing: Research articles," *Wirel. Commun. Mob. Comput.*, vol. 4, no. 8, pp. 881–902, Dec. 2004.

[8] R. Sherwood, G. Gibb, K.-K. Yap, G. Apenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer." in *Tech. Rep. OPENFLOWTR-2009-1*. OpenFlowSwitch.org, 2009.

[9] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: taking control of the enterprise," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 1–12, Aug. 2007.

[10] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with OpenSketch," in *USENIX conference on Networked Systems Design and Implementation (NSDI)*, 2013, pp. 29–42.

[11] K. Chintalapudi, A. Padmanabha Iyer, and V. N. Padmanabhan, "Indoor localization without the pain," in *ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2010, pp. 173–184.

[12] 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.

[13] A. Ishimori, F. Farias, E. Cerqueira, and A. Abelem, "Control of multiple packet schedulers for improving QoS on OpenFlow/SDN networking," in *Software Defined Networks (EWSDN), 2013 Second European Workshop on*, Oct 2013, pp. 81–86.

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

[15] R. Sherwood, M. Chan, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T.-Y. Huang, P. Kazemian, M. Kobayashi, J. Naous, S. Seetharaman, D. Underhill, T. Yabe, K.-K. Yap, Y. Yiakoumis, H. Zeng, G. Appenzeller, R. Johari, N. McKeown, and G. Parulkar, "Carving research slices out of your production networks with OpenFlow," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 129–130, Jan. 2010.

[16] L. Suresh, J. Schulz-Zander, R. Merz, A. Feldmann, and T. Vazao, "Towards programmable enterprise WLANS with Odin," in *ACM Hot topics in Software Defined Networks (HotSDN)*, 2012, pp. 115–120.

[17] P. Calhoun, M. Montemurro, and D. Stanley, "Control And Provisioning of Wireless Access Points (CAPWAP) Protocol Specification," RFC 5415 (Proposed Standard), Internet Engineering Task Force, March 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5415.txt

[18] P. Dely, J. Vestin, A. Kassler, N. Bayer, H. Einsiedler, and C. Peylo, "CloudMAC: An openflow based architecture for 802.11 MAC layer processing in the cloud," in *Globecom Workshops (GC Wkshps), 2012 IEEE*, Dec 2012, pp. 186–191.

[19] S. Bhaumik, S. P. Chandrabose, M. K. Jataprolu, G. Kumar, A. Muralidhar, P. Polakos, V. Srinivasan, and 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.

[20] X. Jin, L. E. Li, L. Vanbever, and J. Rexford, "SoftCell: Scalable and flexible cellular core network architecture," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '13, 2013, pp. 163–174.

[21] K. Pentikousis, Y. Wang, and W. Hu, "Mobileflow: Toward software-defined mobile networks," *Communications Magazine, IEEE*, vol. 51, no. 7, pp. 44–53, July 2013.

[22] A. Gudipati, D. Perry, L. E. Li, and S. Katti, "SoftRAN: Software defined radio access network," in *ACM Workshop on Hot Topics on Software Defined Networks (HotSDN)*, 2013, pp. 25–30.

[23] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow, "Openvirtex: Make your virtual SDNs programmable," in *ACM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2014, pp. 25–30.

[24] L. Sarakis, T. Zahariadis, H.-C. Leligou, and M. Dohler, "A framework for service provisioning in virtual sensor networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2012, no. 1, 2012.