Future Network and MobileSummit 2012 Conference Proceedings
Paul Cunningham and Miriam Cunningham (Eds)
IIMC International Information Management Corporation, 2012
ISBN: 978-1-905824-29-8

A Modular, Flexible and Virtualizable Framework for IEEE 802.11

Pablo SALVADOR[1,2], Stefano PARIS[3], Claudio PISA[4], Paul PATRAS[5], Yan GRUNENBERGER[6], Xavier PEREZ-COSTA[7], Janusz GOZDECKI[8]
[1]*Institute IMDEA Networks, Leganes, 28918, Spain, Email: josepablo.salvador@imdea.org*
[2]*University Carlos III of Madrid, Leganes, 28918, Spain*
[3]*MobiMESH, Milan, Italy, Email: stefano.paris@mobimesh.it*
[4]*University of Rome "Tor Vergata", Rome, Italy, Email: claudio.pisa@uniroma2.it*
[5]*Hamilton Institute, National University of Ireland Maynooth, Maynooth, Co. Kildare, Ireland, Email: Paul.Patras@nuim.ie*
[6]*Telefonica Research, Barcelona, Spain, Email:yan@tid.es*
[7]*NEC Laboratories Europe, Network Research Division, 69115 Heidelberg, Germany, Email:xavier.perez-costa@neclab.eu*
[8]*AGH University of Science and Technology, Krakow, 30-059, Poland, Email:gozdecki@agh.edu.pl*

Abstract: Wireless networks are extensively deployed due to their low cost and configuration easiness. However, they are not adapted to the new services and applications that are increasingly demanded by users. Current implementation of the IEEE 802.11 specification is supported in hardware devices and software developments, but they do not provide the adaptability that would enhance user experience in next generation networks. In this paper, we present a new wireless framework, based on the one currently supported by the Linux stack: *mac80211*. This new framework, named *mac80211++*, has been tailored to improve MAC features in terms of: (i) **modularity**, by defining different 802.11 MAC services; (ii) **flexibility**, by enabling dynamic configurability of the 802.11 MAC; (iii) **virtualization**, by managing parallel independent 802.11 MACs accessing the same system resources.

Keywords: IEEE 802.11, WLAN, architecture, framework, mac80211, driver

1. Introduction

Wireless networks are a popular technology for Internet access. The evolution of new services and applications require WLANs to rapidly adapt to these modifications. However, such evolution require new amendments in IEEE 802.11 standard [1] with the consequently increase of time to approve them. In addition, these new changes must be adopted by manufacturers by exploiting a new set of devices. Consequently, this process is slow and time-consuming, making standardization much slower than real user demands.

Most of WLAN card manufacturers follow the *SoftMAC* approach, much more flexible compared to the old *FullMAC* solution. *FullMAC* leaves all the control of the MAC layer functions to the card hardware/firmware, whereas *SoftMAC* implements a new set of control MAC primitives at the software level. The framework *mac80211* [2], which is part of the Linux 802.11 stack and depicted in Fig. 1a, provides *SoftMAC* capabilities. Nevertheless, the *mac80211* lacks of modularity and flexibility since it is a monolithic block composed of many sub-modules highly interconnected. Following the FLAVIA [3] paradigm, we propose and develop a preliminary implementation of a new framework, namely *mac80211++*, aimed to specify a solution whose advantages are three-fold: modularity, flexibility and virtualization.

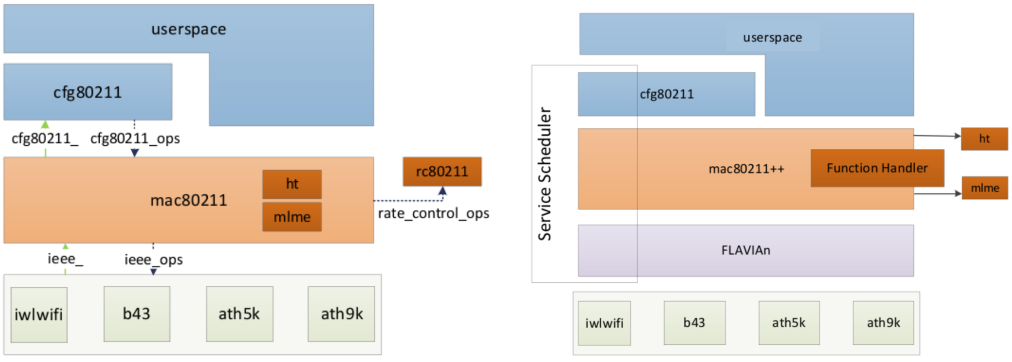
Research supported by the ICT FLAVIA Project, funded by the EU Seventh Framework Programme.

Then, the goals of *mac80211++* are to leverage on the current implementation *mac80211*, widely adopted in 802.11 networks, and to reduce the complexity and shorten the time when introducing changes to the standard. This will boost the implementation of new improved services and reduce the amount of time for these modifications to be commercialized. To this aim, we design a service scheduler and split the *mac80211* framework components. The service scheduler adds and loads new services, flexibilizing the implementation of new modules compared to the standard procedure. Untangling the highly interdependent relations of the *mac80211* components, we pursue to reduce the complexity of the current framework and to foster its modularization, by allowing sub-components being loaded independently.

2. Existing framework

Fig. 1a depicts an overview of the existing Linux 802.11 stack. This stack specifies the framework *mac80211* that enables *SoftMAC*-capable device drivers used for operating with 802.11 hardware. While some of the MAC functionalities are implemented at the hardware level, *mac80211* implements features such as handling several higher-layer components of the MAC, including support for HW/SW crypto, power saving, .11n style aggregation or LED management. The *mac80211* module plays two key roles: (*i*) Wrap the packet incoming from the upper layers and translate them into the 802.11 frame format; (*ii*) Control management operations related to the IEEE 802.11 standard.

A standard wireless driver with Linux wireless capabilities includes some kernel modules and provides interfaces used by user level tools to configure the device behavior, as depicted in Fig. 1a. The main modules defined in the framework are the *mac80211* and the *cfg80211*; these modules are loaded and used by the drivers (e.g., ath5k, ath9k, b43) that are implemented in separate Linux kernel modules.



(a) Overview of the mac80211 framework (b) Overview of the proposed *mac80211++*

Figure 1: Overview of the existing (left-sided) and proposed (right-sided) frameworks.

Bidirectional interfaces are defined among modules as represented in Fig. 1a by the arrows. The exported functions provide a direct interface shown with the solid arrows. The usage of an exported function introduces a dependency in the direction of the arrow (e.g., the driver depends on mac80211). The interface in the other direction is implemented through the registration of callbacks (i.e., function pointers). In Fig. 1a this dependency is represented by dashed arrows and the labels represent the structure containing the function pointers.

The rigidness of the mac80211 is a caveat for developing new services. This framework is a rigid block formed by a set of sub-modules highly interconnected, e.g.: the

MAC layer management entity (*mlme*), the high throughput (*ht*) or the MPDU aggregation (*agg*), as specified in the IEEE 802.11n standard [4]. These parts are defined in dedicated files but not implemented as separated modules, thus preventing any kind of modularization.

3. A new framework: *mac80211++*

Motivated by the rigidness of the *mac80211* framework, we develop a new solution, named *mac80211++*, which aims to overcome it. Fig. 1b depicts the new framework, *mac80211++*, showing the new blocks introduced with respect to the existing framework depicted in Fig. 1a. First, we provide a proof of concept of the modularity of *mac80211++*. Second, we develop and implement a Function Handler and a Service Scheduler that manage the loading and creation of new functions and services respectively, proving flexibility. Third, we describe as well the virtualization support, by adding an overlay layer, FLAVIAn, between the device drivers and the *mac80211++* framework.

3.1 Modularity

The development of *mac80211++* can be considered as a first and relevant step towards the modularization of the wireless component inside the Linux kernel. Some *mac80211* functionalities might be conveniently separated in order to provide the developers with a novel degree of flexibility. Thus, we propose to "break" the monolithic *mac80211* framework and evolve to a new extended and more modular framework. The basic idea is to rely on the definition of well-defined interfaces for those functionalities. To this aim, we follow the rate control module approach replicating its interaction modality. This module depends on *mac80211*, but can be built as an independent module, within the framework or at the driver level, being able to load it at run-time.

Table 1: APIs for *mlme* support ($^* = ieee80211\_$)

Basic mgmt operations	**Event handling functions**	**Power mgmt and saving**
*mgmt_assoc	*sta_rx_queued_mgmt	*dynamic_ps_disable_work
*mgmt_auth	*mlme_notify_scan_complete	*dynamic_ps_enable_work
*mgmt_deauth	*sta_rx_notify	*dynamic_ps_timer
*mgmt_disassoc	*sta_tx_notify	*send_pspoll
*sta_setup_sdata		*recalc_ps
*sta_work		*sta_to_sleep
		*send_nullfunc
		*sta_reset_beacon_monitor
		*sta_reset_conn_monitor
		*sta_restart

Then, by using the same rationale we separate the management (*mlme*) algorithm (STA operation) as well as the support for high throughput (*ht*) from the rest of mac80211 modules. We describe the interfaces for the *mlme* module that we have added through a *mac80211_ops* structure to the *ieee80211_local* structure. Table 1 illustrates the three main categories: (*i*) the specific management part and its setup, (*ii*) the event handling part that includes several notifications, timers and management frames reception coming from the wireless network and (*iii*) functions related to power saving and power management. Fig. 1b illustrates the extensions carried out in the *mac80211* framework, turning it into a more modular framework.

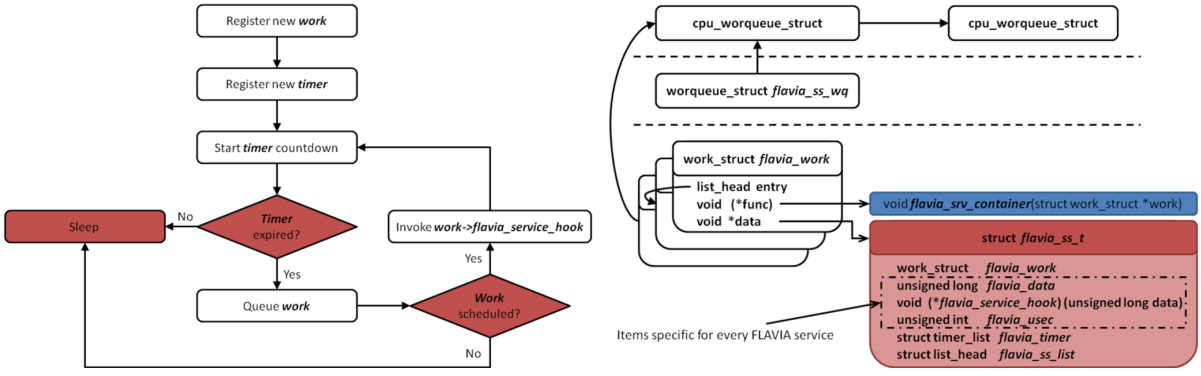
3.2 *Flexibility*

The flexibility provided by *mac80211++* fosters the extension of the basic functionalities defined by the IEEE 802.11 protocol. In particular, our framework permits to implement innovative services and enhanced functions, providing a general yet flexible mechanism to extend the mac80211 framework. To this end, we design and develop two auxiliary kernel modules, namely the *Service Scheduler* and the *Function Handler*, which are liable, respectively, for managing the scheduling of a new service and the registration of the enhanced functions, which are executed at the occurrence of specific events handled by *mac80211++* (e.g., packet reception, packet transmission or channel switching).

Service Scheduler. The Service Scheduler has been designed to provide a simple and standardized mechanism to schedule new services. Through this system, developers can focus only on the implementation of the main service functions, using the Service Scheduler as a mean to schedule periodically its execution. The Service Scheduler will run the functions registered by the service during its initialization phase. In addition, to simplify the implementation of a new service, the Service Scheduler architecture improves its maintenance, since the implementation of its internal functions can be improved to support enhanced services, as long as its APIs are not modified. Indeed, it can be easily updated with more sophisticated functionalities to meet the requirements of real-time systems.

Fig. 2(a) illustrates the main steps to register and execute a new service. When a new service is registered, the Service Scheduler creates a new Linux kernel *work* representing the task implementing the deferred service function, and adds it to a dedicated *work-queue* specifically designed to handle all services. When the timer expires, the Linux kernel *work* implementing the service is queued on the *work-queue*, which contains all the tasks that must be executed immediately. The Service Scheduler defines only one Linux kernel thread to extract and activate the *works* implementing the services on the *work-queue*, in order to serialize the management of all the events occurring in a distributed scenario like the channel access.

Once the *work* can be scheduled, the Linux kernel thread, which handles the *work-queue*, invokes an outer function, namely *flavia_srv_container*, which, in turn, executes the function implementing the service (pointed by *flavia_service_hook*), and reschedules the timer to execute the service later.

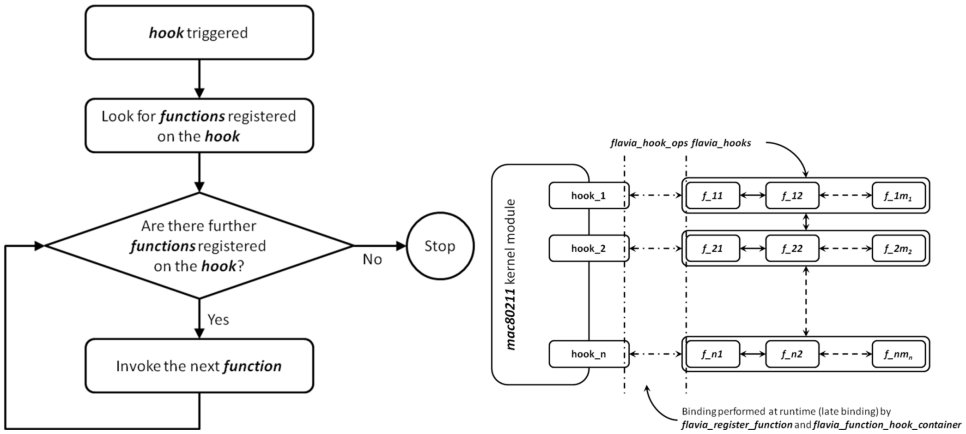


(a) Work Flow (b) Work-Queue

Figure 2: Service Scheduler: Flow chart and work-queue.

Function Handler. The Function Handler (FH) is designed to provide a standardized mechanism to hook the *mac80211++* code (i.e., to add piece of code that acts as glue between any function and the *mac80211* procedures). More specifically, the FH permits

to register a function to any hook added to the *mac80211++* code; thus improving its functionalities with new functions. As depicted in Fig. 3, at the occurrence of a specific event, the Function Handler will call the functions previously registered on that hook. For example, when a new frame is received, the control flow of the *mac80211* code reaches a hook that transfers the control to the FH, which, in turn, invokes the execution of all functions registered on that hook. Note that the function invoked by the Function Handler can register a service or create a new task executed by an independent kernel thread. Therefore, the FH mechanism provides a high level of flexibility to the developers of new functionalities and services.



(a) Work Flow (b) Hooks Double Linked List

Figure 3: Function Handler: Flow-chart describing the main operations performed by the Function Handler and structure used to fulfill the management task.

3.3 Virtualization

While *mac80211++* extends the capabilities of the original *mac80211*, the design of the *mac80211* framework is intrinsically bound to the physical capabilities of the HW advertised by the different device drivers. Past research work [5, 6, 7, 8] has demonstrated that single radio hardware could be virtualized in a way very similar to the virtualization of computational resource in the hardware resources of computer. *mac80211*, and by extension *mac80211++*, brings a logical view to the different wireless interfaces present in the system, but does not offer the proper abstraction necessary for implementing proper virtualization without breaking the existing code base. To overcome this limitation, we design FLAVIAn, an overlay layer that offers virtualization capabilities to *mac80211* and *mac80211++*, while preserving the existing hooks and API.

Therefore, FLAVIAn abstraction is twofold: first, FLAVIAn presents the usual drivers hooks that the traditional *mac80211* is expecting. As well, FLAVIAn proposes the counterpart drop-in replacements to be used in each hardware driver

To ensure appropriate interaction of the *mac80211* stack and the device drivers through the FLAVIAn overlay, a small modification in the driver code is also required in order to reroute the *mac80211* callbacks to the equivalent *ieee80211_flavian_ops* structure specified by the FLAVIAn overlay.

The key functionality covered by these handlers cover frame transmission, enabling/disabling the hardware, configuring Rx filtering or notifying about status of the scanning procedure (start/complete). Similar modifications will be required to support other *mac80211* drivers (e.g., ath9k, b43) with the FLAVIAn overlay, but as we explained above, such changes will involve limited programming effort.

4. Use Cases

This section specifies a representative set of use cases to illustrate the functionality introduced by the *mac80211++* framework.

4.1 Advanced Monitoring

The *Advanced Monitoring Service* (AMS) module provides a passive monitoring service able to measure several parameters related to radio channel conditions, capabilities of neighboring nodes and MAC 802.11 parameters estimation. Each node performs PHY/MAC layer measurements within the time-scale of microseconds. The wireless cards are set to promiscuous mode to ensure a comprehensive view of the current wireless channel conditions. Then, all the measurements are performed within the normal activity of the wireless card and reported periodically. The AMS module supports multiple network interfaces per node. It works on a frame level, meaning that all the frames sent and received by each network interface must be examined by the AMS functions. This imposes high requirements on the AMS module on the effectiveness of the frame analysis (i.e., limited computational power available at the nodes).

The AMS module hooks in the mac80211 module of the Linux kernel are placed in the *ieee80211_rx()* function for the downlink frame path and in the *ieee80211_tx()* function for the uplink frame path. The measurement functions called by the hooks require access to each frame header and frame timing information to discover and calculate a set of parameters per each neighboring station interface, such as: supported rates, SNR, F/BER, RTS_Threshold or number of retransmissions.

For communication with user space the netlink mechanism is used. The application that requires monitoring data from the AMS module sends the command to the receiving function of the AMS module. This command defines the parameters the application requests and the time interval at which results are to be sent to the application.

4.2 SuperSense (SPS)

The virtualization and flexibility features of our proposed framework foster the development of *SuperSense* (SPS), an innovative monitoring service that dynamically analyses the available wireless spectrum using both passive and active techniques to estimate the best network configuration. SPS analyses continuously the available wireless channels to select the set of parameters that provides the best network performance.

The monitoring activity is performed concurrently to the data TX using two virtual interfaces operating over a single physical interface. The virtualization module is liable for scheduling the activities of the different virtual interfaces. In particular, the time spent for data transmission and active monitoring tasks is scheduled according to a time division mechanism implemented using a preemptive weighted round robin policy.

This module sets and manages the total duration of a SPS period and the specific length of the operation modes by introducing a new data structure, the *super-frame*. The duty-cycle of the super-frame, representing the alternation of transmission and monitoring phases along with the time assigned to each activity, is broadcast by the Access Point using a new Information Element (IE) contained in the beacon. The IE contains two main variables indicating the overall duration of the *super-frame* and the time spent to perform the active monitoring. Every *super-frame* always starts with an active monitoring period followed by a transmission period, in which all nodes that belong to the same BSS operate using the same medium access mechanisms (either

CSMA/CA or TDMA) to transmit their data traffic. During an active monitoring frame, only one node is allowed to send probes on the wireless channel in order to estimate actively the quality of the wireless links established with nearby nodes and the interference which might be generated by external sources.

4.3 Power Saving (PS)

The goal of the Power Saving (PS) service module is to enable various power saving algorithms, such as NoA/ASPP [9], to be easily implemented by specifying helpful functional blocks and their interactions with other services or functions.

Thanks to the modularity and flexibility exposed by *mac80211++*, this PS service is easily implemented as a loadable module. For that, we define the following two functions: (*i*) *ps_policy()*, which is registered into the Function Handler. It incorporates the logic of the developed algorithms and stores information of the mechanism(s) in operation and its(their) state; (*ii*) *ps_management()*, which provides management logic to support the PS mechanisms being implemented.

A generic PS scheme might require the ability of triggering sleep/awake events. This action is ultimately performed in HW by setting the proper HW registers accordingly. We then specify a primitive to communicate to the immediately lower layer the notification to execute the chosen event: *drv_ps_notify()*: This is a notification primitive and requires drivers to provide its proper handling. Thus, we push all the "intelligence" to the upper layer, designing this way a hardware-agnostic PS framework. In order to support sleep/awake transitions typically required by power saving algorithms, it is still needed that drivers and firmware support sleep/awake events (issued by the previously mentioned primitives of the PS service).

4.4 Rate Adaptation

Most of the current *mac80211* drivers rely on rate control algorithms provided by the framework. These algorithms are encapsulated in independent kernel modules that are linked to the specific driver once a new device is being loaded. The naming convention of these modules is based on an *rc80211_* prefix, followed by the name of the algorithm. *mac80211* implements two rate adaptation schemes, Minstrel and PID, but also permits the drivers to implement specific rate adaptation mechanisms and register them upon device initialization to notify the *mac80211* framework that rate selection will be handled by the driver itself. One example of such drivers is ath9k for Atheros cards.

Despite the differences of these two approaches, they both use a common mechanism to interface with the *mac80211* framework. Specifically, the *rate_control_ops* callbacks are registered by the rate adaptation module to the framework. These design principles are illustrated in Fig. 4.

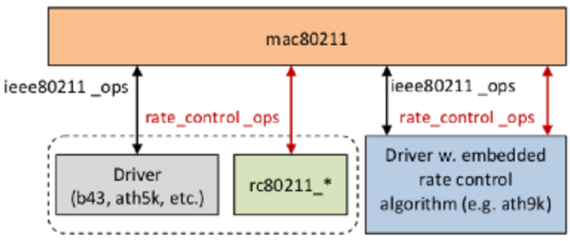


Figure 4: Interfacing Rate Control with mac80211.

Given the platform's modularity and flexibility in allowing the integration of new rate adaptation schemes, we investigate how a collision aware rate control algorithm,

H-RCA [10], could be implemented. This is motivated by the fact that current state-of-the-art algorithms do not distinguish losses due to packet collisions from losses that occur due to noise. The driver incorporating H-RCA takes an approach similar to the rate control algorithms of ath9k. To register the H-RCA algorithm to the *mac80211* framework, the driver is required to invoke *ieee80211_rate_control_register* function passing a reference to a *rate_control_ops* structure, which contains the handlers implemented by the algorithm.

5. Summary & Conclusions

This paper provides the specifications of a high level software architecture that proves the modularity, flexibility and virtualization accomplished by our framework to enhance user experience in wireless networks. Working at that level allows to extend every modification to almost any existing HW.

Our specification has started from an existing framework in Linux, *mac80211*, which has been substantially extended in order to support the aforementioned features. This new framework, namely *mac80211++*, becomes our development platform. Specifically, the extension of the mac80211++ is twofold: (i) We intend to create a modular framework by untangling the existing mac80211, at the present at early stage of development. (ii) We have developed and implemented a Service Handling module that allows loading new services in real-time, based on the mac80211++ framework.

We have identified as candidate modules to be implemented key representative blocks such as SPS and Power Saving. In order to implement the virtualization, a new layer, called FLAVIAn, has been specified between the framework and the wireless drivers, exceeding the bounded capabilities of the driver on which the *mac80211* framework relies.

References

[1] "IEEE Standard for Information Technology-Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks-Specific Requirements - Part 11: Wireless LAN MAC and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999)*.

[2] Linux kernel mac80211 framework for wireless device driver. http://linuxwireless.org/en/developers/Documentation/mac80211.

[3] FLAVIA Project (Flexible Architecture for Internet Access). http://www.ict-flavia.eu/.

[4] "IEEE Standard for Information technology–Telecommunications and information exchange between systems–Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN MAC and PHY Specifications Amendment 5: Enhancements for Higher Throughput," *IEEE Std 802.11n-2009*.

[5] G. Bhanage, D. Vete, I. Seskar, and D. Raychaudhuri, "SplitAP: Leveraging Wireless Network Virtualization for Flexible Sharing of WLANs," in *IEEE GLOBECOM*, pp. 1 –6, dec. 2010.

[6] T. Hamaguchi, T. Komata, T. Nagai, and H. Shigeno, "A Framework of Better Deployment for WLAN Access Point Using Virtualization Technique," in *IEEE WAINA*, pp. 968 –973, Apr. 2010.

[7] L. Xia, S. Kumar, X. Yang, P. Gopalakrishnan, Y. Liu, S. Schoenberg, and X. Guo, "Virtual WiFi: bring virtualization from wired to wireless," in *Proc. of the 7th ACM SIGPLAN/SIGOPS*, VEE '11, (Newport Beach, California, USA), pp. 181–192, 2011.

[8] G. Aljabari and E. Eren, "Virtualization of wireless LAN infrastructures," in *IEEE IDAACS*, vol. 2, pp. 837 –841, Sept. 2011.

[9] D. Camps-Mur, X. Pérez-Costa, and S. Sallent-Ribes, "Designing energy efficient access points with wi-fi direct," *Elsevier Comput. Netw.*, vol. 55, pp. 2838–2855, Sept. 2011.

[10] K. Huang, K. R. Duffy, and D. Malone, "H-RCA: 802.11 Collision-aware Rate Control," *Technical report, Hamilton Institute*, 2011.

