



UNIVERSITÀ DEGLI STUDI DI PALERMO

Dipartimento di Energia, ingegneria dell'Informazione e modelli
Matematici (DEIM)

Dottorato in Ingegneria Elettronica e delle Telecomunicazioni

TESI DI DOTTORATO

Supporting Code Mobility and dynamic reconfigurations over the Wireless MAC Processor prototype

Settore scientifico disciplinare ING-INF/03

Autore:

Fabrizio GIULIANO

Tutor:

Ing. Ilenia TINNIRELLO

Coordinatore:

Ing. Prof. Giovanni Garbo

XXIV CICLO ANNO ACCADEMICO 2012-2013

DOTTORATO



Abstract

Mobile networks for Internet Access are a fundamental segment of Internet access networks, where resource optimization are really critical because of the limited bandwidth availability. While traditionally resource optimizations have been focused on high efficient modulation and coding schemes, to be dynamically tuned according to the wireless channel and interference conditions, it has also been shown how medium access schemes can have a significant impact on the network performance according to the application and networking scenarios.

This thesis work proposes an architectural solution for supporting Medium Access Control (MAC) reconfigurations in terms of dynamic programming and code mobility. Since the MAC protocol is usually implemented in firmware/hardware (being constrained to very strict reaction times and to the rules of a specific standard), our solution is based on a different wireless card architecture, called Wireless MAC Processor (WMP), where *standard protocols* are replaced by *standard programming interfaces*.

The control architecture developed in this thesis exploits this novel *behavioral model* of wireless cards for extending the network intelligence and enabling each node to be remotely reprogrammed by means a so called “MAC Program”, i.e. a software element that defines the description of a MAC protocol. This programmable protocol can be remotely injected and executed on running network devices allowing on-the-fly MAC reconfigurations.

This work aim to obtain a formal description of the a software defined wireless network requirements and define a mechanism for a reliable MAC program code mobility throw the network elements, transparently to the upper-level and supervised by a global control logic that optimizes the radio resource usage; it extends a single protocol paradigm implementation to a programmable protocol abstraction and redefines the overall wireless network view with support for cognitive adaptation mechanisms. The envisioned solutions have been supported by real experiments running on different WMP prototypes , showing the benefits given by a medium control infrastructure which is dynamic, message-oriented and reconfigurable.

Acknowledgements

This work has been supported in part by the EU projects FP7-257263 (FLAVIA) and FP7-258301 (CABIN-CREW).

A special thanks to my Advisor Ing. Ilenia Tinnirello, the professor Giuseppe Bianchi and colleagues Domenico Garlisi, Pierluigi Gallo, Stefano Mangione and Francesco Gringoli.

Contents

Acknowledgements	ii
List of Figures	vi
List of Tables	viii
Abbreviations	ix
1 Introduction	1
1.1 Programmable wireless platforms	2
1.1.1 Code Mobility	3
1.2 Work content description	3
1.2.1 Methodology	3
2 Network Programming	5
2.1 Introduction	5
2.2 Programmable Network	5
2.2.1 Programmable Network Model	6
Data Plane	6
Control Plane	6
2.3 Active Network	6
2.3.1 Framework	7
2.3.1.1 Programmable switches	7
Discrete	7
Integrate - Capsules	7
2.3.2 Programming Model	8
2.4 Software Defined Network - SDN	8
2.5 OpenFlow	10
2.5.0.1 Programmability	11
Header Fields	11
Counters	11
Actions	11
2.5.0.2 Communication between Controller and Switch	11
Controller-To-Switch	11
Asynchronous	12
Symmetric	12
2.5.1 OpenFlow Components	12

2.5.1.1	OpenFlow Switches	12
2.5.1.2	Controller	12
2.6	SDN in Wireles Networks	13
2.6.1	Interference management centralization	14
2.6.2	Self-Organizing networks	15
2.6.3	Session and mobility management	15
3	Medium Access Control Flexibility	17
3.1	Programmable WLAN systems	17
3.2	Overlay Solutions	19
3.3	Dedicated Platforms	19
3.4	MAC programmability with Finite State Machine	20
4	Wireless MAC Processor	22
4.1	Introduction	22
4.2	Architecture	23
4.2.1	MAC Engine	23
4.2.1.1	Instruction set: Actions, Events, Conditions	24
4.2.1.2	MAC abstraction layers	24
4.2.2	MAC Program	25
4.3	WMP-Editor	25
5	MAClet	28
5.1	Introduction	28
5.2	MAClet Control Architecture	28
5.3	MAClets	29
5.3.1	MAClet Distribution Protocol	30
5.3.2	MAClet Synchronization	31
6	MAClet Control Framework	33
6.1	Introduction	33
6.2	MAClet SDN modeling	35
6.3	Architecture	37
6.3.1	Programmable Wireless Nodes and Policies	37
6.3.2	Control System	38
6.3.3	Control Messages and Procedures	38
6.3.4	MAClet Manager	41
6.3.5	MAClet Manager as a Controller	42
6.4	OMF for MAClet Controller	42
7	Experiments and Evaluations	44
7.1	Introduction	44
7.2	Virtualization	44
7.2.1	Scenario Description	44
7.2.2	MAC Virtualization and Synchronization	45
7.2.3	Performance Evaluation	46
7.3	Cognitive Network MAClet switching	47
7.4	Cognitive Channel Selection	49

7.4.1	Experiment setup and evaluation	50
8	Experimental Contribution for vehicular environment networks	52
8.1	Introduction	52
8.2	VANET standards	53
8.2.1	802.11p/WAVE	54
8.2.1.1	802.11p - Physical Level	54
8.2.1.2	802.11p - MAC Layer	54
8.2.2	Motivation	55
8.2.3	Metodologies	55
8.2.4	results evaluation	56
9	Conclusion and future work	60
A	MAClet Manager	61
A.1	maclet-manager options list	61
A.2	Stand-alone Operation Mode	62
A.2.1	Bytecode injection and activation	62
A.2.2	Delayed Bytecode switching	63
A.3	Client-Server Operation Mode	64
A.3.1	Forge template frame operation	67
A.3.2	Collect data	68
B	Hardware Equipment	69
B.1	Introduction	69
B.2	WLAN 802.11 NIC - Broadcom BCM4311 - BCM4318	69
B.3	Wireless Nodes, PC Engines Alix Board	72
B.4	USRP	72
B.4.1	ADC/DAC section	73
B.4.2	The FPGA	74
B.4.3	Usage Example	75
	Bibliography	76

List of Figures

2.1	Network Operating System Distributed State Abstraction	10
2.2	OpenFlow switch Controller-Switch connection	10
2.3	OpenFlow switch matching flow	11
3.1	WLAN MAC architectures: full-MAC vs soft-MAC.	18
3.2	Userspace kernel firmware interaction in GNU/Linux system	19
3.3	a MAC program description with FSM	21
4.1	Internal architecture of the Wireless MAC processor	23
4.2	bytecode binary implementation	26
4.3	WMP-Editor Layout	27
5.1	Architecture for MAClet support: extended WMP and external MAClet Control.	29
5.2	Messages of the MAClet Distribution Protocol: an example.	30
5.3	MAClet MO-message format	30
6.1	MAClet Control framework and OpenFlow complementary	33
6.2	A simple use case of the MAClet distribution system in a multi-vendor scenario.	34
6.3	MAClet Control framework architecture and decoupling between control and data planes	36
6.4	Architecture detail about Policy Control (a) and the Control and Data Planes (b)	39
6.5	MAClet Distribution Protocol: message exchange and MAClet synchronization	40
6.6	wmplib for MAClet/WMP interaction	41
6.7	OMF System Architecture	43
7.1	two operators share the same physical Access Point by providing two different access services to their clients (fixed-rate and best-effort).	45
7.2	An experimental trace of network virtualization: operator A and operator B use the channel in different time intervals with independent access schemes (TDM and DCF).	45
7.3	Resource repartition between two different operators using different access rules (TDM and DCF).	47
7.4	Cognitive Loop DirectLink	48
7.5	MAClet Activations	49
7.6	Channel hopping experiment setup	50
7.7	Channel hopping evaluation	51

8.1	802.11p channels	54
8.2	DSRC STACK	55
8.3	Interarrival time - 6Mbps - 10MHz	57
8.4	Coverage Percent - 6Mbps - 10MHz	58
8.5	Busy Channel Monitoring	59
B.1	BCM43xx NICs	70
B.2	Broadcom BCM 43xx Block Diagram	71
B.3	Alix 2D2	72
B.4	USRP1 USRP2 frontend and USRP2 block diagram	73
B.5	USRP Tracker interactive tool	75

List of Tables

2.1	OpenFlow Table Entry	11
2.2	OpenFlow Controllers Comparison	13
3.1	MAC programs expressed as Extended Finite State Machines: Wireless MAC Processor	21
5.1	WMP Commands to be locally or remotely invoked	29
6.1	MAClet Controller messages	39
B.1	Broadcom BCM4311 Specification	70

Abbreviations

AM	A ggregate M anager
EC	E speriment C ontroller
ED	E speriment D efinition
FSM	F inite S tate M achine
IFS	I nter F rame S pace
MC	M AClet C ontroller
MLME	M edia A ccess C ontrol (M AC) S ub L ayer M anagement E ntity
MM	M AClet M anager
NIC	N etwork I nterface C ard
WMP	W ireless M AC P rocessor
SM	S tate M achine
SDN	S oftware D efined N etwork
WMPE	W ireless M AC P rocessor E ngine

Chapter 1

Introduction

In recent years Wireless network for Internet Access has become one of the most important infrastructure in communication technologies and IT application. The impressive growth of features and bandwidth capabilities in 3G/4G cellular networks and the pervasive distribution of WiFi access networks for business and home internet access are the proof that Wireless based access network is today the *de facto* internet access for users. However not all network technologies currently in use are able to scale with a large amount of users and not all medium access schemes can be exploited without a knowledge of the dynamic environment; for that reason the research community and industry try to address the problem of medium access limitations defining new standards and specific solutions pay particular attention to free frequencies wireless local network technologies.

One of the most famous WLAN standard is IEEE 802.11. It was born just for cable replacement in local area networks with limited management functionalities and data rates. The evolution of 802.11 (e.g. 802.11e, 802.11sm 802.11n, 802.11ac, etc.) brought the definition of new standards with highest performance and spectrum usability. The definition of current new standards can clearly address a set of weakness and give more features but very often novel application scenarios or networking topologies require the introduction of updates. Indeed, wireless network networks suffer of a number of issues of channel access and reconfigurability of networks elements, that strictly depend on the operating network conditions and interference. For this reason an interesting research path can be focused on the designing of a new paradigm for network reconfigurability and optimization, according to which nodes can change dynamically not only the modulation and coding schemes, but also their behavior in establishing wireless links and accessing the wireless medium. Obviously, such a reconfigurability requires that that all nodes can share information about channel state, are updated in a consistent way and follow some common (global or distributed) optimization algorithms. The programmability of future Internet is a current interest research topic, where the definition of programmable interfaces and abstractions do not only involve the wireless access network. For example, these concepts have been largely debated in the context of switched networks, where Openflow [52] has obtained a large consensus for trading-off reconfiguration capabilities of switching network elements and usability of the interfaces.

The main purpose of this work is to design a dynamic network reconfiguration model and define a cognitive paradigms for Medium access control adaptation. For this scope, a prototype architecture for flexible and programming of MAC protocol called Wireless

MAC Processor[38] has been used. This framework is a powerful element for designing a wireless network where nodes can define, modify and run MAC protocol software based implementations called "MAC programs". The final goal is identifying a suitable programming interface and programming language to code the MAC protocol logic in a compact program that can be carried in one control message and configure remotely the MAC layer of the network devices by simply disseminating these special messages. In this way it is possible designing a smart access network that can solve dynamically a number of medium access control issues such as channel quality fluctuations, traffic load, radio spectrum optimizations and cognitive algorithm implementations.

This work starts from the analysis of the most important proposals for software defined network in switched networks studying research contributions and solutions, analyzing the benefit that this improvement give to the network flexibility; after that has been conducted an analysis of the design requirements needed to obtain software defined network paradigm for wireless access network, and finally has been developed a prototype of a communication system using IEEE 802.11 commodity devices implementing the WMP API. Finally, we analyzed typical usecases of medium access control reconfigurability in different scenarios and evaluated the results.

1.1 Programmable wireless platforms

Current GNU/Linux network module system are designed using SoftMAC[4] paradigm, with this approach part of MAC Layer Management Entity (MLME) is managed in software by the Operating System. MLME include Authenticate, Deauthenticate, Associate, Disassociate, Reassociate, Beacon, Probe, Timing Synchronization Function (TSF). All the functions that are non time-critical can be developed by software and leave in the lower levels the implementation of the time-constrained protocol operations, for example the acknowledge signaling. The project Linux Wireless [3] define an implementation on Linux Kernel side of the IEEE 802.11 common management entities that interacts with driver modules of each NIC and implements softMAC paradigm, moreover several low level MAC/PHY parameters such as congestion windows, frequency, antenna selection, bandwidth or rate can be configured by software. However, this level of flexibility is not enough to bring into real world optimization solutions which require a fine parameter tuning for each frame or for control frames such as beacon or acknowledge frames. To overcome this limitation the research community has developed custom programmable wireless platforms, typically revolving around an FPGA or DSP core. Modern platforms, such as WARP [19] are stand-alone software defined radio boards equipped with fast and large FPGAs. These solutions allow the implementing of whole wireless protocol stack, from the level of signals to that of frame payloads, they support full MAC layer customization and cross-layer designs. More recently, custom MAC programmability was made possible also on commodity card, thanks to the disclosure of a (simplified) open source firmware [10] for a brand name card. A complete control of source code of PHY/MAC entities permits any modification, but the main purpose of a dynamic auto-configurable network is that the MAC logic can be defined as a finite state machine that a generic program executor is able to run. The Wireless MAC Processor project [38] proposed a prototype architecture developed with this specific purpose when has been implemented a MAC Engine on firmware level that is able to execute a MAC program in terms of finite state machine. A set of APIs is given to the developer that can compose its MAC and inject into the card enabling the new features on the fly. In dynamic network

reconfiguration this approach allow to design a communication architecture that can move this code throw the network transforming a fixed wireless network into a flexible environment that choose with cognitive paradigm the best medium access scheme.

1.1.1 Code Mobility

Despite the hype in the midst of the nineties [32], the application of active networking principles to the wireless domain has lagged behind. In the vision of [67], adaptations, envisioned in terms of selection of PHY functionalities (spectrum access, modulation, and coding) were expected to leverage software radio technologies. But proposed wireless active networking frameworks [32] have mainly addressed issues at layers higher than low-MAC/PHY (e.g., QoS, network topology adaptation, mobility, ad hoc network formation, etc). The interest for code mobility, also embedded in in-band data packets (capsules, as per [70]), has more recently emerged in the wireless sensor networks arena. Indeed, in large sensor networks, code mobility may be the only possibility for upgrading the sensors' behavior, given that physical access to the nodes may not be viable. But, again, programmability has been restricted to higher layers, and for tasks such as changes in the monitoring functionalities or in the application operation [59]. Especially in the sensor network field, several issues concerning code distribution protocols[42, 45] and architectures [50] have been considered. Obviously, the programmability requirements for wireless local networks have some differences from the sensor network ones. Sensor nodes deployed in the same network are usually homogeneous, with the same Tiny OS and hardware. A binary code image can be moved from a node to another in active messages (natively supported by TinyOS). Albeit not strictly necessary, byte-code interpreters[44] may significantly improve efficiency of code distribution, i.e. for giving an high-level virtual code representations which significantly reduces the code length and/or facilitate incremental updates. All the above referred solutions limit programmability to network, transport and application protocols, and assume that the lower stack dealing with medium access and single-hop communications is not modifiable [44].

1.2 Work content description

This work is divided the following chapters: Chapter 2 describes the solutions adopted in Internet Network for reconfigurability of its elements such as router, switch, firewall; in chapter 3 a discussion about the motivation of a MAC reconfigurability and a design analysis for software flexibility requirements; in chapter 4 a description of main features of WMP prototype used for protocol code reconfigurability and in chapter 5 is presented MAClet, a solution designed and implemented for a wireless network architecture that support code mobility; in chapter 6 the study of the MAClet Control Plane design, in chapter 7 a description of the usecases that adopt WMP/MAClet software defined network architecture.

1.2.1 Methodology

The design of MAClets and MAClets Controller started from the decoupling of the requirements of a complex network scenario in terms of physical and network elements,

the defining of a programmable MAC infrastructure, the analysis of the benefits of a MAC layer reconfigurability and the definition of a model for each requirement. The modeling is the most important aspect because it can make the difference between a solution that can be pervasive and one solution that can live only in particular conditions. That's the hard aspect of this work and in general not all the solution seems to be actually the optimal one, but with a solid background model and with a modular system is easier find the weak element and replace with a stronger one.

Chapter 2

Network Programming

2.1 Introduction

This chapter is focused on a conceptual and technical analysis on current solutions for network programmability: we describe the main features of Programmable Networks, the motivations, and the main concepts envisioned to rethink the traditional telecommunication networks and extend its features in a reliable and flexible context.

The timeline of Programmable network evolution has experienced different steps and contributions. In 1995 OPENSIG [24, 32] was proposed with the purpose of making ATM networks more flexible; GSMP (General Switch Management Protocol)[16] has been standardized to establish connections between the switches, manage switch ports and reserve resources; in 1997 Active Networks[66] have emerged for reconfiguring packet oriented networks by embedding programs in data packets; in 2004 the project 4D proposed a separation between routing logic and routing protocol in network elements [15], while in 2006 was developed Ethane [2], the predecessor of OpenFlow that in 2011 was introduced by ONF foundation[8, 52]. In 2008 With NOX[5], an OpenFlow controller was finally proposed.

In the following sections, we provide a description of the main research work on these technologies and on the impact that these solutions can have for defining programmable mobile access networks.

2.2 Programmable Network

Vendor dependent and hardcoded implementation of control software do not allow a flexible programming of nodes: this is a limitation for innovation because any novel solution requires to be standardized, implemented by vendors and accepted by final users (that need to buy novel network nodes) before it can be deployed in real networking applications. Moreover some implemented features are often proprietary and have a poor level of reconfigurability on users side. In [32] the problem has been studied defining a network modeling and a programming model in network environment, examine the number of solution and aim to define a common way to define a network that is programmable and able to react quickly in terms of reconfigurability and flexibility. The

first issue to address is to identify a strategy to separate the hardware data forwarding rules from the control protocol, and define a software-based paradigm for the control protocols. In this way switch, router or any other network component can be described as a general purpose network element that change its control protocol by changing of control protocol software. A new network component can be defined basically writing a new software program, and by the definition of a common set of APIs it is possible to describe a common language to allow all components to be interoperable. At this point it is crucial to define a model for this new type of network, a network that is not a fixed environment for a specified protocol but a dynamic system that can involve its features by software changing. The envision of a programmable network is that network behavior can quickly change also in runtime, by the definition of a program distribution logic that allows a fast network reconfiguration.

2.2.1 Programmable Network Model

The modeling of a network has a number of complex requirements and the way to give programmability must go through an intensive analysis of the problem with an important abstraction modeling. One of the distinctive characteristics of a programmable network is that, starting from a minimal set of APIs, it is possible to compose several services. A general way to design a programmable network is based on a model that defines data, control and management plane for each layer of internet stack. This general envision can be applicable in every kind of telecommunication networks and the separation between transport and control are highlighted in network architectures. Giving a fine grained programmability of each plane allows to define a reliable system that is able to implement quickly new features and applications. The Data Plane/Control Plane separation is the first step to delegate decision functions to network elements able to aggregate the network information and take the right decision for data forwarding.

Data Plane is the part of the network that carries the traffic and refers to the information that is being transported. In general data plane decides what to do with a packet, analyzes the header and acts following a communication protocol.

Control Plane Defines the rules used by Data Plane. In a traditional network each node has been made with an implementation of control protocol without basically any necessity of modification. This implies predefined network protocol configuration and there's no models for adaptation or dynamic rules redefinition. This paradigm has a weak part: each evolution must provide a complete per-node network modification, in most cases with hardware replacement or low-level code upgrade. A programmable network node does not implement a specific control protocol but defines a programming language that allows to implement in software any kind of protocol starting from a very simple APIs set.

2.3 Active Network

In 1997 [66] introduced the Active Network paradigm, an architecture approach that was born with the main purpose to increase the flexibility of internet networks allowing

a programmability level based on propagation of program code within the packets. This milestone work envisions a network that is able to overcome the issues of a wide area network that suffer of problems for integrating new technologies and standards into the shared network infrastructure, as well as poor performance due to redundant operations at several protocol layers, and difficulties for accommodating new services. Moreover the standardization process of the Internet Protocol can take years to define a new common standard; the new standard must be developed by vendors and after that the entire network must swap from an old technology to the newer one. Changing the protocols means changing the network operations. Several strategies, collectively referred to as active networking, have emerged to address these issues. In the program-based approach, the networking system is a combination of small program elements that give to the network nodes the information for a cognitive decision for several optimizations. The goal is obtaining a network where each new developed feature can be easily downloaded and installed into endpoint nodes and routers. This approach allow a faster pace of innovation bringing a hardcoded network architecture to a Virtualized paradigm.

2.3.1 Framework

Active Network is a network designed with nodes that are able to receive configuration packets and process them using programming directive. Users can send dissecting rules for control packets that can implement new features for each node giving the advantage to modify a traditional hard-coded set of switch and router configuration rules to program-dependent configuration.

2.3.1.1 Programmable switches

A programmable switch is an element that allow dynamic programming of packet dissection and routing rules definitions and allow to analyze data frames and take decision on it. In this approach users can send program messages that inject the routing rules adapting the behavior of nodes based on the traffic type and the program message sent. Active Networks can be designed in discrete or integrate approach: the first separate the programming messaging phase from the data flow, the second aim to integrate programming messages and data into each single packet.

Discrete In this approach program and data are separated and it is defined a set of mechanisms for each type of message. When a program message is sent, the programmable node receives and injects the message into its configuration interface. It will dissect the future data packets with the rules included in the program sent previously. The approach is useful in case of large programs or if is necessary a previous control of the program.

Integrate - Capsules A Capsule is a packet with data that is also a network program. In this solution, all the packets exchanged into the network have an embedded program and all nodes in the network have a common set of capabilities for reading and executing the programs: each packet contain the program with the dissection rules to be executed for it. This approach is an example of code mobility concept and shows how a packet

fine programmable network can define a framework for a generalized subset of usecases and applications.

2.3.2 Programming Model

In Active networks arise the important aspect of the network programs requirements: a network program is a communication element that must be a common resource for all nodes of the network, it must define a set of common instructions that can be interpreted by each node and it must define a common description of resources handling. The main requirements of a network program are:

- mobility - the program can be run on different platforms;
- safety - handling program resource access;
- efficiency - programs must not compromise the network performance.

The network program is defined as a script that can be executed on-the-fly when the packet is received, when the internal logic of active switches is able capture the embedded code and execute it. The important aspect is in term of real time execution with null or at least small compiling time. The scripts use a small and simplified set primitives that are the same for all nodes, so a fundamental element of the framework is the APIs set that define data analysis commands and may include decision rules, condition and events.

2.4 Software Defined Network - SDN

The Software Defined Network paradigm is based on the definition of a network made of nodes with an high level of programmability. A general SDN must define an abstraction model for the programmability requirements arisen from the network analysis. As for Active Networks, the first element to analyze in SDN is the separation between Data Plane and Control Plane; the second element is about the centralization of Control Plane and consequently the abstraction models to be defined. In case of Internet, three fundamentals programming issues have been defined to be solved for control plane design. In traditional networks most control operations (e.g. the routing) are implemented by means of a distributed control protocol and each network element has to implement the same protocol to obtain network interoperability. The main issue is that when a node is programmed the only way to modify, update or improve the network features is reprogram each node from scratch. For these reason it is important to define Control Plane where is possible to keep the control of network nodes and interact with them. To this purpose, it is required to define a reliable model so that each programming command can be forwarded with a standard communication protocol; second the physical network environment must not be a constrain for the network element control and finally each device must be configured in transparent way. Summarizing a software defined network needs the follow control requirements:

1. Operate within a given network-level protocol;

2. Operate without communication guarantees - a distributed system with arbitrary delays and drops;
3. Compute the configuration of each physical device - A common way to introduce the programs in networks elements in this way every physical device is consider an equivalent element.

The Control Plane for a network system has several complexity aspects that must be managed, but, if we want to obtain a synergistic implementation we must extract simplicity with the right abstractions. For this three requirements the Control plane must define the following abstraction models:

1. Forwarding;
2. Distributed State;
3. Detailed configuration.

Forwarding

This abstraction model must give an interface that shields the upper layers from the embedded low-level implementation of the forwarding mechanisms. Network elements handled by Controller are developed to be able to interpret the same instructions. OpenFlow, for example, defines a so called *Matching Table* that contain the Control rules; the definition of the Forwarding model depends on the type of SDN designed. For switched networks, adopting a matching table seems to be a good solution but in many other SDNs a forwarding model based on a flowchart or a state machine can be more effective. The motivation is inside the programming level that a Controller gives to the network nodes.

Distributed state

Network must have a global view. The controller gets a global view of the network state and responds with the configuration of each device. The way to obtain a global view of the network is the Network Operating Systems (NOS). NOS is a distribute system that create the network view, communicates with the forwarding element of the network and communicate the control directive to the network elements. Controller is placed on top of NOS and views the network as a graph.

Detailed configuration

The abstraction model for network devices configuration consist of a **Control Program**: a control program is simply a function that gets inputs from the network view and distributes the configuration for each node.

An interesting overview of these concepts is provided in OpenFlow v1.0.0[\[11\]](#), which describes the programmability approach proposed for software-defining switching nodes.

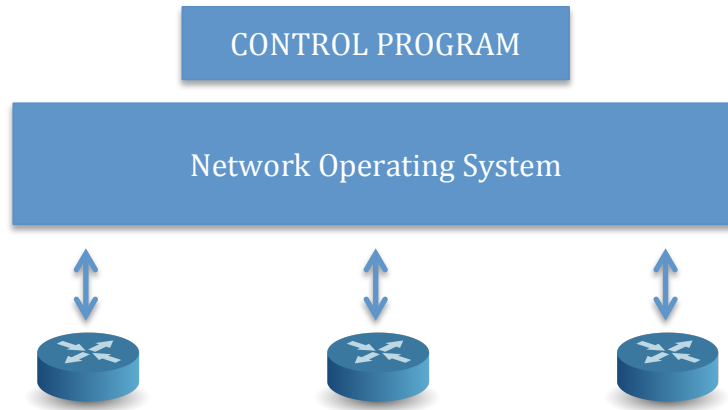


FIGURE 2.1: Network Operating System Distributed State Abstraction

2.5 OpenFlow

One of the most important well-know SDN-inspired architectures is OpenFlow[52], whose features are summarized in this section.

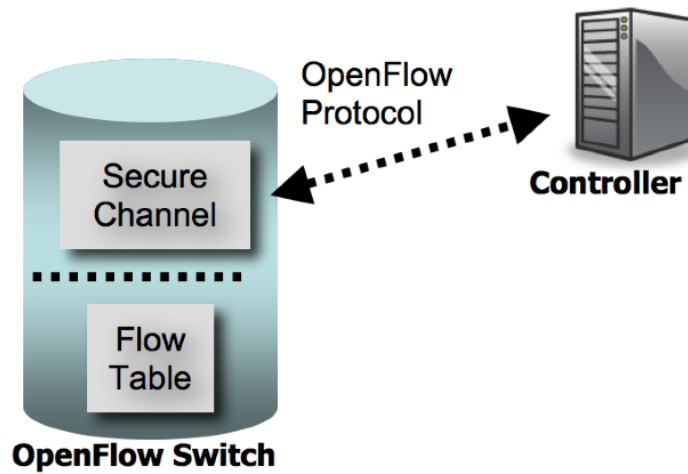


FIGURE 2.2: OpenFlow switch Controller-Switch connection

The architecture decouples Control and Data Planes and define two network primary elements: the *Controller* and the *Switches*. The *Controller* and the *Switches* exchange information using the so called *OpenFlow protocol* which define:

- *Controller-to-Switch messages*;
- *Flow Table structure*;
- *Flow Match structure*;
- *Flow Action structures*.

Each Switch loads a flow table that defines a rule, an action and a set of statistics information; for each packet received it dissects the packet analyzing the flow table and takes the decision executing the action. In figure 2.3 it is shown the packet matching algorithm for an OpenFlow switch.

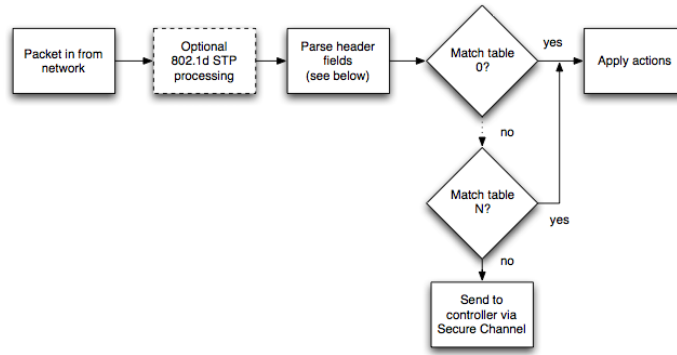


FIGURE 2.3: OpenFlow switch matching flow

2.5.0.1 Programmability

In OpenFlow the programmability level is driven by the Controller that injects to the Switch the Flow Table. Table inspection logic of OpenFlow is logically assimilable to a state machine that loops in a single state and checks a single event: a packet reception. When a Packet is received, OpenFlow checks throw a list of conditions defined into the flow table and executes one of the actions described. Table 2.1 shows the description of a single entry of a Flow Table.

Header Fields	Counters	Actions
---------------	----------	---------

TABLE 2.1: OpenFlow Table Entry

Header Fields is a list that contain the header fields value that the switch must check for matching.

Counters Contains the statistics information per-table, per-flow, per-port, per-queue. These information are updated at each packet arrival.

Actions Define how the switch must handle the matching packets. Actions are classified in "required" or "optional"; basically each action defines the forward, enqueue, drop, modify-field operation executed for each packet matching.

2.5.0.2 Communication between Controller and Switch

The communication between the Controller and Switch is obtained throw an interface called *secure channel*. OpenFlow protocol defines three types of messages: *controller-to-switch*, *asynchronous* and *symmetric*.

Controller-To-Switch These type of messages are initiated by the controller and may or may not require response from the switch.

Asynchronous Message sent from the Switch to the Controller without soliciting. With these types of messages, the Switch updates some information to the Controller.

Symmetric These are bidirectional messages without solicitation.

2.5.1 OpenFlow Components

The OpenFlow components are hardware switches and software to assist OpenFlow protocol implementation and give maintenance, debugging and monitoring tools.

2.5.1.1 OpenFlow Switches

The impressive OpenFlow growth brought an important commercial contribution. In fact there are a number of business devices able to work with OpenFlow; HP ProCurve, NEC IP8800, Pronto with Indigo or Pica8 firmware are an example of models that implements OpenFlow. There are also Software/Test Switches for OpenFlow: NetFPGA, OpenWRT, PCEngine WiFi AP and OpenVSwitch.

2.5.1.2 Controller

Also for controller there are several implementation in different language. The first OpenFlow controller is **NOX** that supports OpenFlow v1.0.0. It is written in C++ and define an API. NOX has two separate lines of development:

- NOX-Classic is the well-known line of development that has been available under the GPL since 2009. It contains support for Python and C++ and a bunch of applications. However, this line of development is deprecated.
- NOX only contains support for C++, has fewer applications than NOX-Classic, but it is much faster and has a much cleaner codebase. Today developers are working in this new codebase. For python prototyping, there is also POX. NOX defines the classes for OpenFlow protocol objects, network state consistency manager, table sending, table dropping and so on. The main problem of NOX is the difficult to be deployed and a large number of users have trouble for building and running NOX in their environment.

NOX core only provides very low-level methods for interfacing with the network. All higher-level functions and events are created and provided by network applications (components). A component is just an encapsulation of controller functionalities. For example, routing is implemented as a component within NOX. Any function which requires routing must declare it as a dependency to ensure its availability at runtime. Developer recently move to another implementation called **POX**. It is implemented in python and use both interpreter cpython or PyPy. POX give more features and allow an easier prototyping work. POX also define a list components of stock components that define the most important switch implementations. This framework provide a set of APIs to implement customized components. in [\[14\]](#) the wiki page of POX install/setup guide,

the list of the stock components and a guide for developing custom components. NOX and POX are not the only Controller available for OpenFlow, many opensource and non-opensource products have been developed. In table 2.2 a summary of most important OpenFlow Controllers.

Control- lers	Lang- uages	Docs	Open- Source	Institu- tions	Multi- thread	Notes
NOX	C++/ Python	Good	Yes	Nicira Networks	Yes	Widely used
Maestro	Java	Fair	Yes	Rice Uni- versity	Yes	No Ref.
Trema	C/ Ruby	Poor	Yes	•	Yes	No Ref.
Beacon	Java	Good	Yes	Stanford	Yes	Very Used
Helios	C	?	No	NEC	•	No Ref.
BigSwitch*	Java	?	No	BigSwitch	•	Production Netowrk
SNAC**	C++/ Python	?	No	Nicira Networks	•	Production Netowrk
POX	Python	Good	Yes	Nicira Networks	NO	Under Develop

TABLE 2.2: OpenFlow Controllers Comparison

2.6 SDN in Wireles Networks

The most critical (and most distinct) aspects of wireless networks are interference/radio resource management and mobility management, which obviously work on different complexity scales when we consider cellular or local access networks. While for cellular networks interference control and mobility are natively provided in standard functions and configuration interfaces available to operators, they are often add-on functionalities for WLANs like IEEE 802.11 – a technology born as unmanaged wireless access for small and ad hoc deployments. However, both cellular and WLAN landscapes are evolving faster than standardization [46]. On one hand, femtocells deployments and the flexibility of modern WCDMA/OFDMA technologies make cellular networks less planned. Especially in the content of femtocells, it is impractical – or impossible if the device is installed by the customer! – for the operator to manually tune the installation and soft parameters of all its base stations. On the other hand, dense Access Points do not work well by keeping the network control completely distributed, because it is often impossible to prevent critical interference problems. In both cases, recent literature has shown that enabling or introducing a network-wide resource control can result in a performance improvement. Nevertheless, most of these insights have not migrated to real world deployments because they are not natively supported by existing wireless nodes and standards.

2.6.1 Interference management centralization

In LTE an emerging approach for improving the network capacity is centralizing some resource control operations [39] to overcome the limitations of local decisions, which are myopic by necessity, and also obtaining some deployment benefits, mostly from hardware resource pooling. Networks based on this architecture are called cloud radio access networks (C-RAN). C-RANS can be seen as an effective implementation of the idea of multi-cell processing or network MIMO because the digitalized I/Q radio signals of different antennas are processed together. On the basis of this processing, the central controller can decide whether the antennas should be used for boosting signals (combining diversity to boost signals for delay-sensitive applications) or for spatial multiplexing (multiple parallel transmissions). The promise of such approach includes energy efficiency, load balancing, and capacity improvement due to the cooperative processing of the signals received and transmitted by distributed base stations, which in turns require an efficient data sharing strategy between the base stations and the cloud [62]. In [49] a hybrid solution is proposed, where centralized and distributed baseband configuration and processing decisions coexist according to the bandwidth availability between the base stations and the cloud. Indeed, when the capacities of the backhaul links are infinite (or sufficiently large), the uplink joint processing problem becomes that of a multiple-access channel, and the downlink becomes a broadcast channel for which the capacity regions can be easily computed. However, in practical scenarios, the sampled waveforms and the interference measurements need to be carefully compressed for dealing with the finite capacity of the backhauls [60]. Moreover, for scalability reasons, the optimization can work on a cluster of cells rather than on the whole network [62]. Also for WLANs, major vendors, e.g., Cisco, Aruba, and Meru, have realized that a centralized network structure can be exploited for optimizing network performance. For example, DenseAP [55], and Trantor [56], have proposed to centralize various radio control functions, including rate selection across client-AP links, transmit power settings, interference-aware associations, load balancing, etc., that can significantly improve the utilization of the radio resources. In these solutions, client nodes have a very limited intelligence: they collect measurements describing their local view of the network (e.g. the observed air-time) and act according to the configuration commands sent by a controller to APs and stations (through the serving AP). Some authors have pushed these principles further and propose to centralize the link access control, which has been traditionally considered part of the data plane. For example, [63] shows that the global view of the network can be exploited for avoiding that APs suffer from hidden or exposed transmissions. The idea is to opportunistically schedule the forwarding of downlink data frames to the APs in order to avoid simultaneous transmissions in case of hidden APs, and synchronizing exposed APs by means of pre-determined backoff values. A centralized architecture for simplifying and optimizing the management of network infrastructures with multiple APs has also been standardized by the IETF with the CAPWAP protocol [9], which—in principle—is technology agnostic and requires specific “bindings” for each considered access standard (so far, there is only the binding for 802.11). Services offered by CAPWAP concern security and registration, but also the centralized control of radio resources and configuration of Wireless Termination Points (i.e. the APs in the CAPWAP terminology) for improving network performance [43], [26]. Radio configuration is expressed in terms of management information base elements included in the standard, such as the operating channel, the transmission power, and the channel sensing mode, but also the beacon interval, the contention parameters and the retry limits used by the medium access scheme. Device configuration is expressed in terms of hardware (radio interfaces)

and firmware selection. Configuration decisions are taken by a central controller called Access Controller (AC).

2.6.2 Self-Organizing networks

An alternative paradigm to centralization, is dealing with the increased scale, complexity and heterogeneity of emerging networks by means of self-configuration. Self-configuration of wireless points of access within a single technology has been heavily investigated within the LTE standard for LTE eNodeB [37], [35] focusing on address and cell identity assignment, and for Wi-Fi APs, focusing, among others, on IP address configuration [23], channel assignment and AP selection [40], [25]. State of the art self-optimization schemes for LTE networks concentrate mainly on handover threshold optimization [47], the neighbour relation problem [51] and dynamic subcarrier assignment [29]. Automatic configuration and self-optimization for heterogeneous networks has not been extensively studied, although some initial proposals appeared in the literature (e.g., VET [65]). Self-organizing networks are already proven to be effective on theoretical basis (e.g., [61], [68]). However, practical realizations under realistic assumptions, especially under imperfect channel state estimation, constrained communication between base stations, and small cells (which make the distributed solutions complex and less reliable) are largely lacking.

2.6.3 Session and mobility management

Current solutions for session and mobility management in cellular networks centralize some critical functions which introduces scalability problems. For example, monitoring, access control, and quality-of-service functionality are centralized at the packet gateway, which may be a performance and cost bottleneck for the whole system [46], especially when a significant amount of traffic is directed to other nodes within the cellular domain. Some mechanisms proposed to address this include the possibility to delegate access and service control to a remote controller (external to the gateway); dynamically adapting the traffic paths within the cellular core network and introducing a fine-grained control in the internal routing enables several interesting possibilities for the operators, such as balancing the traffic between different service gateways, differentiating customer services, improving resource utilization (and consequently reducing CAPEX costs). The fine-grained control of per-session resource allocations is especially important for some applications which are expected to become major contributors of the overall traffic, i.e., video streaming [41], or important drivers of the network economics, i.e., Machine Type Communications (MTC) [69]. Optimization for both categories would require flow-based routing, billing, and Quality of Service (QoS) configuration, over both the core and the access network. With the appearance and success of Software Defined Networking [41], [69], and OpenFlow [52] for controlling the traffic flows within a wired network by means of a central controller and simple interfaces for programming the network switches, various research efforts have proposed similar approaches for wireless networks. In this direction, OpenRoads [71] tries to support experimentation on different mobility management solutions by using OpenFlow for configuring the data-path within the core network, SNMP for configuring the network devices (both the switches and the wireless access points) and signalling critical network events, and an OpenFlow controller

[5, 13] for building mobility management applications. The approach enlightens the importance of wireless device configuration, because parameters like transmit power can directly impact on the data-path performance, but it limits such a configuration to a few PHY-related parameters.

Chapter 3

Medium Access Control Flexibility

More than 20 years have elapsed since the establishment, in 1990, of the IEEE 802.11 Wireless Local Area Network committee. Initially foreseen as a technology for replacing Ethernet cables with wireless connectivity, IEEE 802.11 has been severely challenged by the highly heterogeneous needs emerged in the last two decades. Indeed, the original 802.11 CSMA/CA Medium Access Control (MAC) has shown significant shortcomings when facing the breakthrough rate improvements made available by the latest PHY enhancements (802.11n, 802.11ac), as well as when applied to scenarios and contexts such as ad hoc and mesh networks, vehicular environments, directional antennas, quality of service support, real time media streaming support, multi-channel operation, dynamic spectrum access, and many others.

Actually, the WLAN research community has found effective and ingenious solutions for adapting the 802.11 MAC operation to these new challenges. However, most of the proposed MAC modifications do not comply with the 802.11 standard MAC operation. In the best case, i.e., when the required MAC amendments are endorsed by some 802.11 standardization task groups, several years may elapse before they become available in commercial cards/devices. More frequently, when the promoted MAC amendments are either deemed out of the standard task groups' scope, or mandate a “way too significant” departure from the native CSMA/CA MAC operation, their real world deployment is very unlikely, especially when they require changes in time-critical operations natively implemented in the network interface card.[38].

3.1 Programmable WLAN systems

The ability to modify the operation of commodity WLAN systems goes along with the availability of public-domain open-source 802.11 MAC protocol code. Besides the significant expertise required to modify existing code, a further deployment barrier for many appealing MAC extensions consists in the limited extent to which software changes may affect the device operation. Indeed, early 802.11 devices were designed according to a *full-MAC* approach. The MAC layer was almost entirely implemented in the card hardware/firmware, and programmability of the relevant drivers (when provided as open

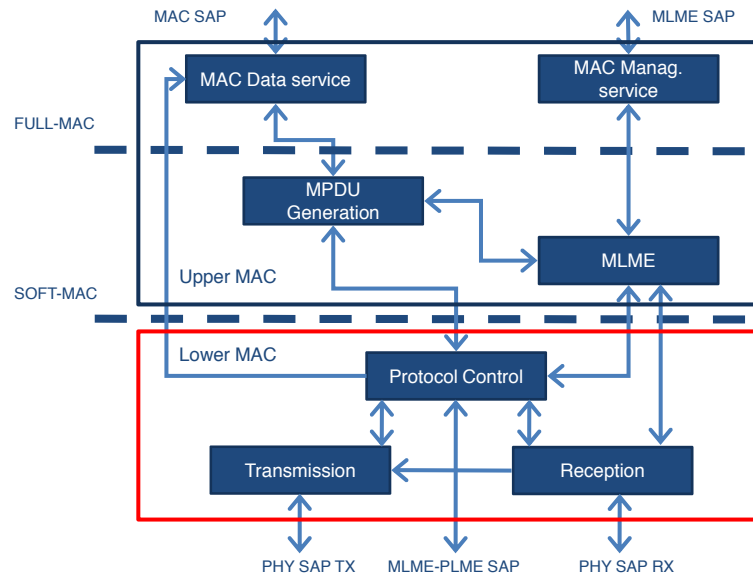


FIGURE 3.1: WLAN MAC architectures: full-MAC vs soft-MAC.

source) involved a marginal set of functionalities. The flexibility of commodity WLAN cards has significantly improved since a number of vendors (including Intel, Ralink, Realtek, Atheros, Broadcom), started to exploit an innovative *soft-MAC* [20] design, transferring to the host processor non-time-critical MAC layer functionalities (figure 3.1). Still, even in the *soft-MAC* case, the “Lower MAC”, comprising crucial subsystems such as transmission, reception and protocol control, remains hard-coded in the card. Although some chipsets (e.g. from Atheros and Broadcom) permit the tuning of selected MAC parameters (such as contention windows) via registers, more substantial MAC operation changes require access to the firmware code.

Newest GNU/Linux System implements mac80211, a framework which driver developers can use to write drivers for SoftMAC wireless devices. SoftMAC devices allow for a finer control of the hardware, allowing for 802.11 frame management to be done in software for them, for both parsing and generation of 802.11 wireless frames. Most 802.11 devices today tend to be of this type, FullMAC devices have become scarce. mac80211 [4] implements the cfg80211 callbacks for SoftMAC devices, mac80211 then depends on cfg80211 for both registration to the networking subsystem and for configuration. Configuration is handled by cfg80211 both through nl80211 and wireless extensions. In mac80211 the MLME is done in the kernel for station mode (STA) and in userspace for AP mode (hostapd). If you have new userspace utilities which support nl80211 you do not need wireless-extensions to support a mac80211 device. In figure 3.2 the stack description of a recent network modules stack for GNU/Linux Operating Systems.

Up to now, no vendor has to date released an open source firmware, and the only available public-domain code is OpenFWWF [10], a recently released simplified DCF firmware implementation for Broadcom/AirForce chipsets. However, OpenFWWF extensions require reimplementing of large portions of assembly code, thus making it usable only by experts.



FIGURE 3.2: Userspace kernel firmware interacioint in GNU/Linux system

3.2 Overlay Solutions

A significant effort has been spent on the development of overlay software modules. Solutions such as the Overlay MAC Project [7], MultiMAC [31], FlexMAC [48], Soft-TDMAC [58], etc, do exploit firmware configuration registers and some driver hacks for building quite advanced MAC programming interface (for instance, MultiMAC permits to override the frame format, disable ACKs, RTS/CTS, virtual carrier sense, disable transmission backoff, etc). Even if notable implementations of custom MAC protocols, including TDMA-like ones, have been demonstrated, overlay approaches cannot get rid of some intrinsic limitations. Their scalability may be impaired by the need to overlap and duplicate similar functionalities at different layers; they remain constrained by the basic programming interface made available by the driver; and their limited ability to accurately control the card's timing prevents to deploy features such as programmable management of frame replies and handshakes, precise scheduling of medium access times, fine-grained radio tuning control, etc.

3.3 Dedicated Platforms

The shift from commodity wireless cards to dedicated wireless platforms permits to push programmability much farther, although the beneficiaries remain mainly confined within the research community - real world deployment of costly and/or bulky platforms being unlikely.

Early platforms such as RUNIC [21] and CalRadio [1] re-implemented the 802.11 MAC protocol stack on, respectively, a Xilinx FPGA and a Texas DSP, interfaced to a commercial PHY-only Intersil 802.11b chip. As such, they permitted arbitrary MAC modifications, but protocol reconfiguration required a deep knowledge of the platforms and could only be done offline by recompiling the modified C code.

Software defined radio (SDR) platforms, such as GNURadio [17] and USRP [18], overcome the dependency on a specific PHY interface and permit to develop full-custom MAC/PHY cross-layer protocols. A large amount of work focuses on means to improve the slow SDR performance. On one side, solutions such as SORA [64] achieve a throughput comparable to commodity 802.11 hardware by distributing computation on multiple cores and by relying on sophisticated optimizations, as well as on an efficient radio control board. However, the software complexity makes protocol stack modifications not easy, as any update implies a redesign of the software block repartitions to multiple CPU cores. On the other side, platforms such as WARP [19] and AirBlue [57] improve performance by delegating most processing functions to the FPGA Hardware, meanwhile retaining the ability to closely control such functions via, e.g., registration of interrupt handlers, hardware triggers, read/write of hardware registers, etc. In the case of AirBlue, a modular organization coupled with careful design choices permits relatively easy modifications, changes in a module not affecting the others.

3.4 MAC programmability with Finite State Machine

Despite the above discussed advances in programmable wireless systems, the belief that wireless access programmability should go well beyond the ability to just "hack" firmware/software code implementing a pre-established MAC protocol stack, and should rather be *designed into* the MAC stack architecture.

MAC protocols are well suited to be described in terms of Finite State Machines. Indeed, they are used in the formal appendices of the 802.11 (and many other) standard. In particular, eXtended Finite State Machines (XFSM) are a generalization of the finite state machine model and permit to conveniently control the *actions* performed by the MAC protocol as a consequence of the occurrence of *events* and *conditions* on configuration registers. An XFSM is formally specified through an abstract 7-tuple (S, I, O, D, F, U, T) : the meaning of such symbolic states and the correspondence with the MAC terminology above introduced is summarized in Table 3.1 (configuration commands being a special case of *actions*, devised to update registry status). Thus, a *MAC program* can be simply considered as a table listing all possible state transition relations. Note that the number and meaning of the set of *protocol states* is specified by the programmer. By formally describing, per each protocol state, which events and conditions do trigger a state transition, and by associating actions and configuration commands to each state transition, the programmer may access the available hardware primitives, and enforce a desired MAC behavior within the radio hardware. Since the configuration memory is not explicitly represented in the state space, XFSMs allow to model complex protocols with relatively simple transitions and limited state space. A generic Architecture able to interpret this paradigm can execute not one but several MAC protocol starting from a common set of commands. In figure 3.3 an example of a MAC state machine.

The Wireless MAC Protocol Architecture[38] is a prototype architecture that supports a set of Medium Access Control "commands" which can be run-time composed (programmed) through software-defined state machines, thus providing the desired MAC protocol operation. An architectural description of WMP and a discussion about the features on code mobility will be explainend in chatper 4.

XFSM formal notation		meaning
S	symbolic states	MAC protocol states
I	input symbols	Events
O	output symbols	MAC actions
D	n-dimensional linear space $D_1 \times \dots \times D_n$	all possible settings of n configuration registers
F	set of enabling functions $f_i : D \rightarrow \{0, 1\}$	Conditions to be verified on the configuration registers
U	set of update functions $u_i : D \rightarrow D$	Configuration commands, update registers' content
T	transition relation $T : S \times F \times I \rightarrow S \times U \times O$	Target state, actions and configuration commands associated to each transition

TABLE 3.1: MAC programs expressed as Extended Finite State Machines: Wireless MAC Processor

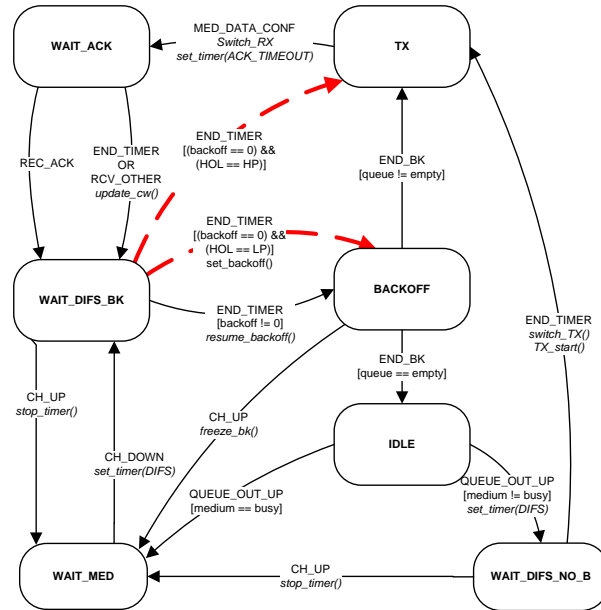


FIGURE 3.3: a MAC program description with FSM

Chapter 4

Wireless MAC Processor

4.1 Introduction

In what follows we briefly review the WMP main concepts [38] and anticipate some discussion on extensions for supporting code switching. Indeed, a pre-requirement of *any* wireless active MAC framework is the ability to support customized MAC operation on general-purpose wireless devices, and the possibility to switch to a desired MAC protocol logic described through suitably formal languages and application programming interfaces.

The **Wireless MAC Processor (WMP)** is an architecture platform devised to run a wireless MAC program defined in terms of a **Finite State Machine (FSM)**. It has been shown, in fact, that MAC protocols can be described in terms of state machines made of three main elements: actions, events and conditions. In the WMP case, actions are commands for the radio hardware, such as transmit a frame, set a timer, and switch to a different frequency channel. Events include hardware interrupts such as channel up/down signals, indication of reception of specific frame types, expiration of timers and so on. Conditions are boolean expressions evaluated on internal configuration registers that can either explicitly updated by actions, or implicitly updated by events. Some registers store general MAC layer information (like the current radio channel or the power level), or more specific MAC variables (like the contention window value and the backoff parameter). Starting from an initial (default) state, the WMP waits for events which trigger state transitions. The actual transition can be enabled or disabled by verifying a boolean condition, while an action on the hardware system (i.e. on the transceiver) can be performed before completing the transition to the new state.

For these reasons, the WMP differs from off-the-shelf wireless NICs powered with their “vanilla” code: while the latter are tied to a specific MAC protocol (i.e., IEEE 802.11), the WMP architecture can run generic FSM, hence it can implement users’ designed MAC programs. On the basis of a pre-defined (hardware-dependent) set of actions, events and conditions which represent the platform API, a MAC programmer can easily compose different channel operations into a MAC program and execute it on the WMP.

4.2 Architecture

The Wireless MAC Processor architecture somewhat mimics the organization of ordinary computing systems, where programmability is accomplished by specifying i) an adequate *instruction set* which permit to perform elementary tasks on a machine; ii) a *programming language* which conveys multiple instructions (suitably assembled to implement a desired behavior or algorithm) to the machine, and iii) a Central Processing Unit (CPU), which executes such program inside the machine, by fetching and invoking instructions, updating relevant registers, and so on.

The wireless MAC processor has been conceived as a CPU specialized for handling hardware/PHY events and actions by executing Extended Finite State Machines (XFSMs). State machines are very effective in modeling the behavior of sequential control operations, and most MAC protocols are formally described in terms of state machines. Figure 4.1 shows the internal architecture of the WMP, which includes five main components:

- an execution engine, running the provided XFSMs;
- a memory block including both data and program memory space;
- an interruption block passing the signals coming from the hardware to the execution engine;
- a set of operations which can be invoked by the execution engine, which include logic, arithmetic and flow control operations plus specialized MAC operations;
- a set of registers for saving system state parameters.

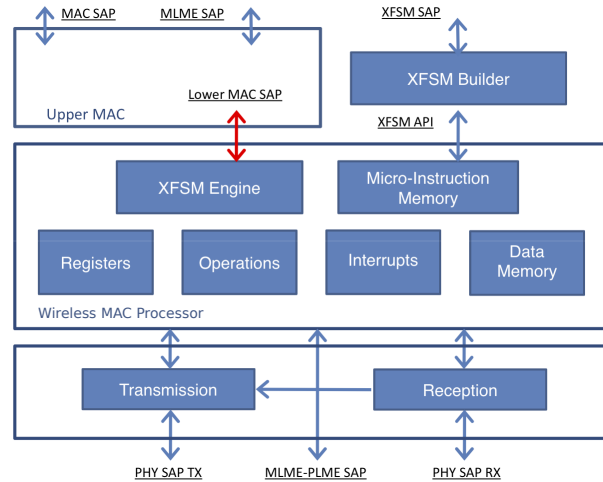


FIGURE 4.1: Internal architecture of the Wireless MAC processor

4.2.1 MAC Engine

The **MAC Engine**, analogous to the control unit of a microprocessor, is the core of the architecture. It performs the tasks of fetching the MAC program, translating it into logical operations and basic actions and scheduling actions on the hardware.

The engine is in charge to execute the **MAC program**, written by the developer as an FSM, and (dynamically) loaded in the WMP micro-instruction memory from the host PC. Starting from the current state, the engine waits for *events* (input signals), then it verifies whether optional triggering *conditions* are verified, in which case it executes the *action* and the state change.

4.2.1.1 Instruction set: Actions, Events, Conditions

A breakdown analysis of MAC protocols reveals that they are well described in terms of three types of elementary building blocks: *actions*, *events* and *conditions*.

Actions are commands acting on the radio hardware. In addition to ordinary arithmetic, logic, and memory related operations, dedicated actions implement atomic MAC functions such as transmit a frame, set a timer, build an header field, switch to a different frequency channel, etc. Actions are *not* meant to be programmable. As the instruction set of an ordinary CPU, they are provided by the hardware vendor. The set of actions may be extended at will by the device vendor, and complex actions may be considered, so as actions not necessarily restricting to MAC primitives (e.g. perform a PHY encoding/decoding).

Events include hardware interrupts such as channel up/down signals, indication of reception of specific frame types, expiration of timers, signals conveyed from the higher layers such as a queued packet, and so on. As in the case of actions, also the list of supported events is a-priori provided by the hardware design.

Conditions are boolean expressions evaluated on internal *configuration registers*. These registers are either explicitly updated by actions, or implicitly updated by events. Some registers are dedicated to store general MAC layer information (such as channel used, power level, queue length), frame related information (source or destination address, frame size, etc), or more specific MAC parameters (contention window, backoff parameters, etc - used to achieve a more compact protocol description in case of specific MAC designs such as CSMA-based ones).

Actions, events, and registers on which conditions may be set, form the application programming interface exposed to third party programmers. This API is implemented (in principle) once-for-all, meaning that programs may *use* such building blocks to compose a desired operation, but have no mean to modify them. However, this API was not envisioned for supporting code mobility. For instance, we could not enforce conditions to control the switching between a previously running MAC code and a newly uploaded one. Thus, we needed to extend the WMP *internals* to implement an extended API accounting for new actions, events and registers, tailored to dynamic code management.

4.2.1.2 MAC abstraction layers

MACs defined for the WMP can be considered following two abstraction layers: a textual one, where everything is described using text expressions, and a graphical one, where the state machine is described through a practical graph based approach. The former representation is the **Bytecode**, a text file that can be either written at hand by users, or automatically generated by the **WMP-Editor**, a graphical tool that can be used

to build the latter representaion. This tool helps users composing new FSM, elements can be, in fact, connected together thanks to a straightforward drag and grop interface. At this point a compiler converts it into a binary format, the **Binary Bytecode**, so that the resulting file can be executed by the WMP. The Binary Bytecode is ready to be pushed inside the WMP executing memory area. **Bytecode Manager** is the management tool used to interface users to WMP.

4.2.2 MAC Program

Figure 4.2 shows an example WMP bytecode where we can recognize the initial state descriptor and the transition table. The table is coded by: i) a list of transition lists, and ii) a list of states represented by the pointer to the relative transition list. When the length of the transition list of a given state is higher than 8 transitions, an explicit list delimiter is used (namely, *FFFF*). Otherwise, the list length is specified by the last 3 bits of the pointer. For example, the state in the second position of the list (whose symbolic label 01 corresponds to the position index) points to the transition list coded from the third byte of the table and ends at the occurrence of the first *FFFF* delimiter. As evident from the figure, the code is very compact (only 544 bytes). Each Transition is composed by the follows elements:

- State address: give the physical address of the actual state.
- Event/Condition Parameter, and a Action Parameter, these are parameter useful for extended functionalities of each Event/Condition/Action.
- Event/condition identificator: is the address of the event that must be verified.
- Action idetificator: is the address of teh condition that must be verified
- Next state identificator: is the address of the next state

bytecode Injectinion and activation are performed by an application software called Bytecode Manager. This tool has been developed to obtain an interface between users and MAC Engine. The main operations of bytecode manager consist of loading/injectin of bytecode. Bytecode Manager loads loaded a file with a text-based XFSM description and injects it into the memory of the NIC. The other operation consist of the Bytecode activation, while the bytecode is loaded it can be activated by a direct communication to the MAC Engine. In current implementation WMP define two memory areas for bytecode fetching.

4.3 WMP-Editor

(WMP-Editor) is a graphical tool that represents state machine programs as transition graphs. Users can edit WMP graphically, adding new states and transitions and customizing the WMP behavior working on its atomic elements, namely conditions, actions and events as introduced in section 4.2.1.1. The same tool can be used as a compiler to translate the transition graph into a Bytecode.

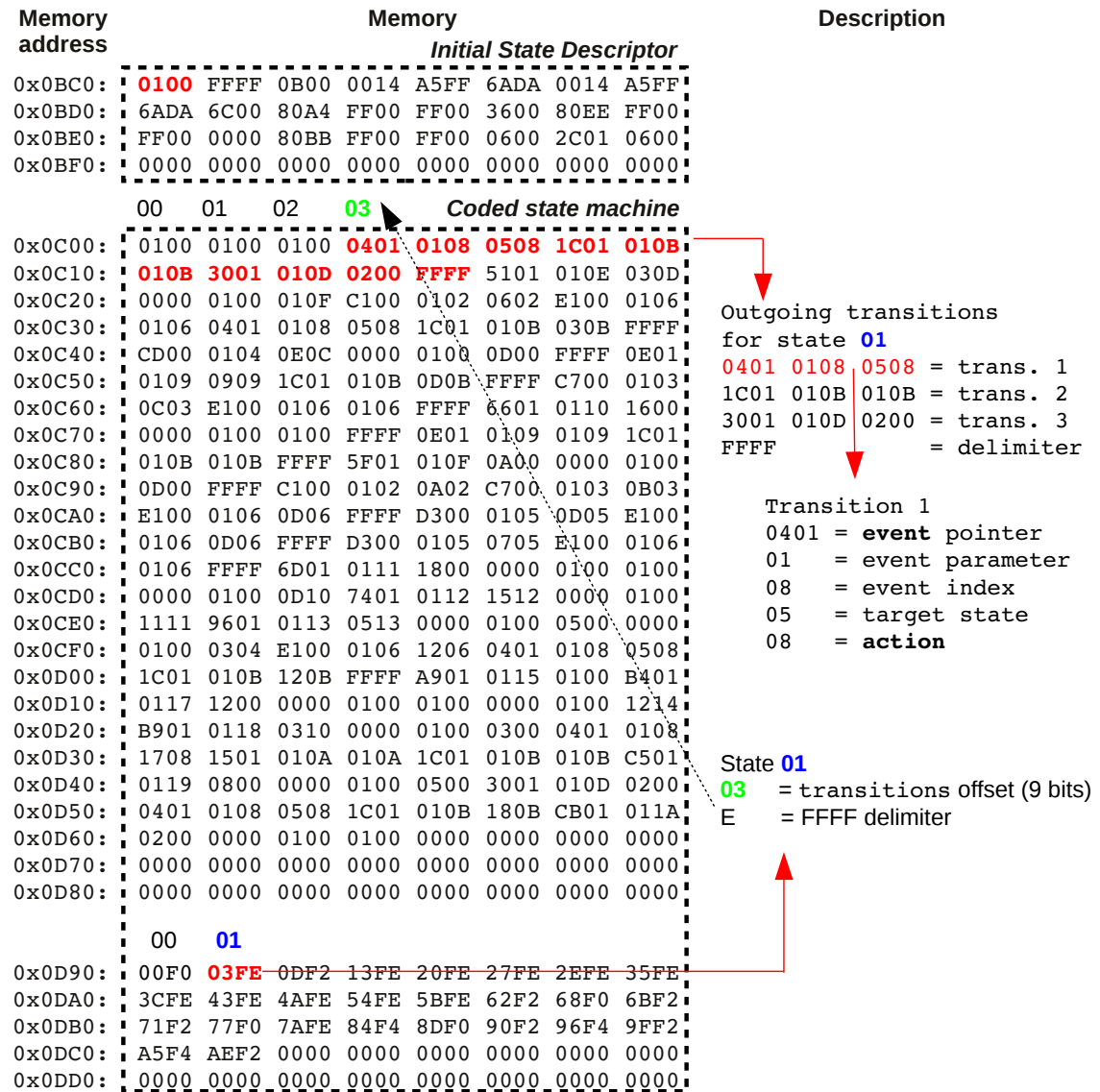


FIGURE 4.2: bytecode binary implementation

The basic Layout of WMP-Editor (see Figure 4.3) is an all-in-one window organized into three main frames: on left, global parameters of the state machine; the middle frame where the graphical state machine is composed; and the bottom frame that hosts the user interface for creating and modifying program states and transitions. In details:

- **Parameters Frame** It includes all the environment variables of the state machine. Two types of parameters can be distinguished: **General** and **Enhanced** parameters. General Parameters set the value of the WMP configuration registers (e.g. the hardware register specifying the operating channel) and the initial state from which the MAC-Engine starts the execution. The Enhanced Parameters allow to specify other program parameters, not strictly related to the default configuration registers, such as MAC addresses to be used for filtering purposes, a channel hopping sequence, a time slot interval, a pre-defined constant backoff value, and so on.

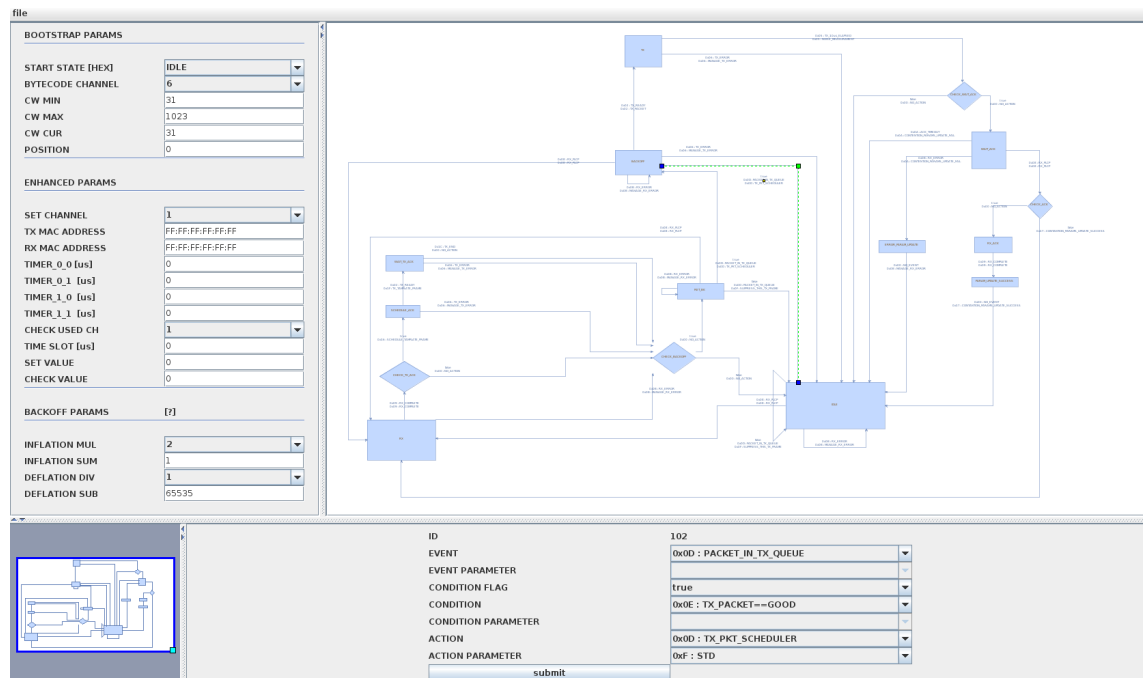


FIGURE 4.3: WMP-Editor Layout

- Machine Building Frame** It displays the state blocks and the transitions defined by the programmer. WMP-Editor uses a simple right-click pop-up to add and edit state blocks and transitions.
- User API Frame** It is the bottom area of WMP-Editor where programmers modify the properties of state blocks, condition blocks (if explicitly included in the machine representation), and transition elements, by specifying events, conditions and actions for each transition from the set of available API.

Chapter 5

MAClet

5.1 Introduction

This chapter introduces MAClets, software programs uploaded and executed on-demand over wireless cards, and devised to change the card's real-time medium access control operation. MAClets permit seamless reconfiguration of the MAC stack, so as to adapt it to mutated context and spectrum conditions and perform tailored performance optimization hardly accountable by an once-for-all protocol stack design. Following traditional active networking principles, MAClets can be directly conveyed within data packets and executed on hard-coded devices acting as virtual MAC machines. Indeed, rather than executing a pre-defined protocol, we envision a new architecture for wireless cards based on a protocol interpreter (enabling code portability) and a powerful API.

5.2 MAClet Control Architecture

In this section we describe how low-level MAC functionalities can be *encapsulated* in a MAClet and transferred from a node to another of the network by exploiting the WMP API and a MAClet distribution protocol.

Figure 5.1 shows the envisioned system: the control architecture is a *pure* software architecture, running at the application level, that interacts with the enriched WMP by means of an open control API. This approach has several advantages. First, the selection of the MAC protocol can be based not only on low-level performance parameters (such as the link quality, the interference conditions, etc.), but also on high-level context estimates, including the application requirements, the network topology, the user preferences, and so on. Second, the code distribution model (handshaking mechanisms, peer-to-peer code sharing, server-client uploading, etc.) is completely independent on the underlying programmable interface, thus allowing full flexibility and a wide range of applications for the same platform. Moreover, the communication delays between the host and the card have a minimum impact on the MAClet Control, since the dynamics of the networks (which require MAC protocol customizations) are reasonably much slower than the processing delay due to an application-level decision module.

More into details, the architecture is based on four main components: the WMP control interface, the MAClet manager, the MAClet Controller and the MAClet repository. The

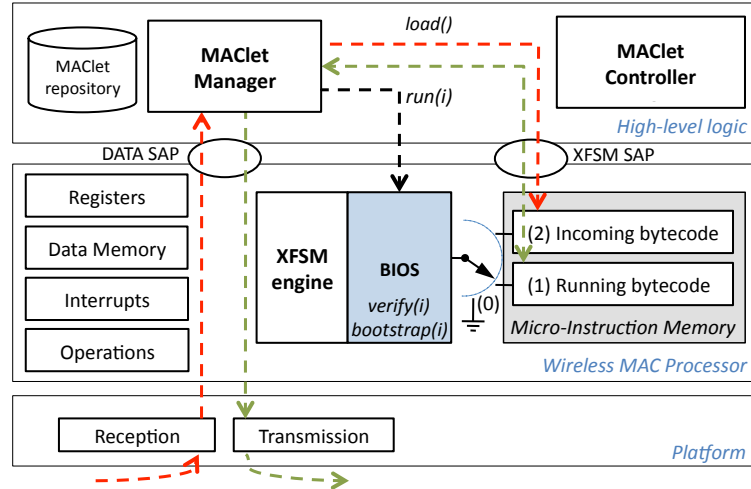


FIGURE 5.1: Architecture for MAClet support: extended WMP and external MAClet Control.

WMP Control Interface	
load i	load a MAC program on memory slot i
run i, e	activate MAC program on slot i (asynch. or at the event e)
verify i	recognize trusted code by means of an hard-coded signature computation
switch $i, j, t, a/r$	add or remove a switching transition t from the slot i to j

TABLE 5.1: WMP Commands to be locally or remotely invoked

WMP control interface is the interface to the hard-coded device, through which new MAC state machines and switching conditions are loaded on the card, as summarized in table 5.1. The MAClet manager is responsible of receiving/transmitting MAClets and MAClet protocol messages, enabling the loading on the card, and programming MAC reconfigurations. The MAClet Controller is the intelligent part of the system, dealing with the network-level configuration decisions in a centralized way (e.g. at the Access Point only), or in a distributed way (e.g. by involving multiple cooperating controllers, sharing both the monitored data and the available MAClet tables).

5.3 MAClets

A key component of our architecture is the code transport unit, i.e. the MAClet. A MAClet is a *coded state machine* with an *initial state description* to be fed on the wireless device.

Being n_s the number of symbolic protocol states and n_e the number of events revealed by the device, a common approach for coding XFSMs is using a $n_s \times n_e$ table, where at each location (i, j) is stored the state transition when event j is received at state i . A transition is defined by a triplet (a, c, s) , specifying the action label a , the enabling condition label c and the target state label s . As each state generally reacts to a number of input events much lower than the total input number, the state machine coding can be

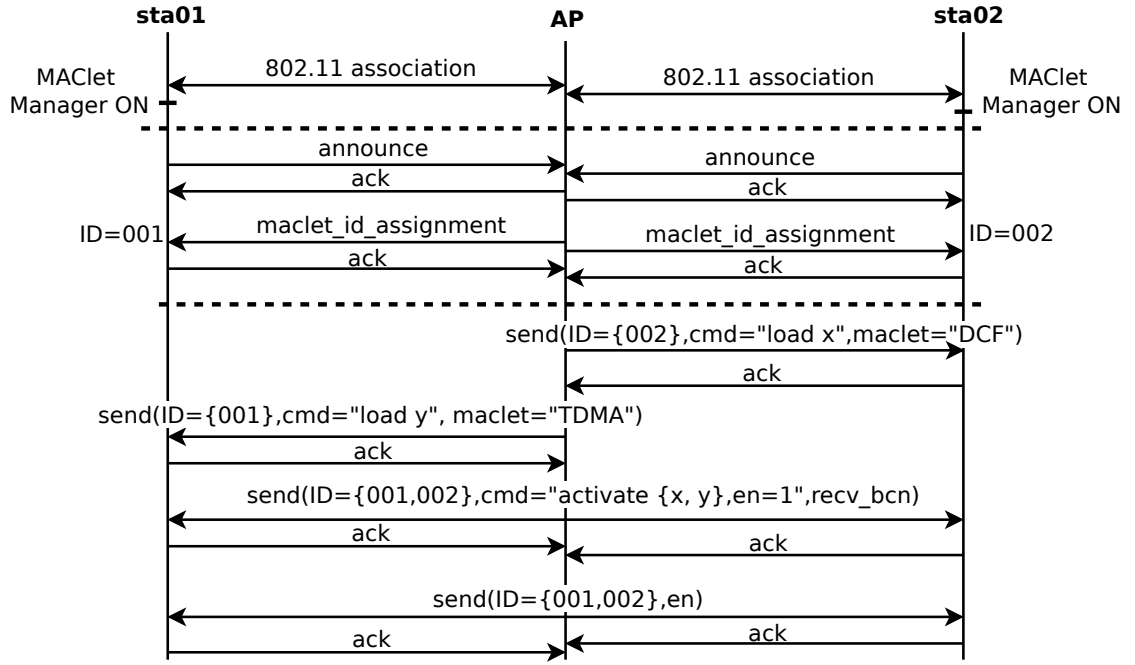


FIGURE 5.2: Messages of the MAClet Distribution Protocol: an example.

TYPE=OP	ID 1	ID 2	...	ID n	CMD	MACLET	PARAMS	activation_event
---------	---------	---------	-----	---------	-----	--------	--------	------------------

FIGURE 5.3: MAClet MO-message format

optimized by skipping null-transitions. The initial state (from which the state machine has to be run) includes the protocol logic state and the platform configuration registers. For example, according to the API defined in [38], these registers (of equal size) specify the settings of the physical channel, the slot size, the contention window values, the current backoff, the transmission power, the retry limit, generic protocol timers and MAC addresses to be filtered. Optionally, the initial state descriptor can be extended with a signed digest of the MAClet code to be used for verifying trusted code sources¹.

As detailed in what follows, MAClets are transmitted with a special message of the MAClet Distribution Protocol, called MAClet action message.

5.3.1 MAClet Distribution Protocol

MAClets can be propagated in the network by means of a *physical transport network*. This means that nodes can negotiate the activation of a new MAC protocol only if they belong to the same network (on a given channel) and employ a compatible MAC protocol. Standard MAC protocols can assume the role of default *common* protocols to be executed (eventually, on a pre-defined *common* channel) for supporting dynamic reconfigurations. We assume that the default protocol and configuration parameters are pre-loaded in each WMP as a *bios* state machine (e.g. in our implementation, the bios machine is a legacy DCF working on channel 1).

¹Although most of the security issues can be demanded to the MAClet Control, this function can be used by manufacturers for controlling the MAC program origins and limiting or avoiding third-party reconfigurations.

The MAClet Control process runs as a normal distributed application, whose messages are defined by a protocol called MAClet Distribution Protocol. This protocol is responsible of: i) collecting information for estimating the network context; ii) negotiating the network reconfiguration decisions; iii) transporting the MAClets and the relative activation signals; iv) verifying the network consistency after a reconfiguration.

The protocol includes two types of messages: MAClet *management messages* for associating each MAClet manager to a MAClet Controller running the distribution logic and confirming control operations, and MAClet *action messages* for transporting MAClets and remotely invoking the desired WMP control functions. When a new station activates, it tries to associate to an AP (acting as a MAClet Controller) by using the bios state machine. In case of success, the MAClet manager is activated for enabling the reception of AP messages. An announcement message is sent to the AP for notifying the activation of the new MAClet Manager and receiving an identifier. According to its decision logic, the AP is then able to send a specific MAClet action messages to each associated station, to a group of stations or to all the network stations (see figure 5.2).

The MAClet action message comprises the following fields: the list of destination addresses of the relevant MAClet managers, a command to be executed on the addressed WMPs, the MAClet bytecode, the MAClet configuration parameters, and the MAClet activation data. Not all fields are always included in the action messages: for example, it is possible to specify a new set of parameters for a MAClet already loaded on the station without carrying the relevant bytecode.

5.3.2 MAClet Synchronization

Achieving a network-level reconfiguration is obviously much more complicated than working on a single node, because it is necessary introducing some forms of coordination. In particular, the activation of a new MAClet on different nodes could require a common reference signal for avoiding critical inconsistencies (such a temporary use of different transmitting channels) leading to disassociations or other network errors.

The MAClet Control Architecture provides the primitives for programming the desired synchronization and error recovery operations, but the specific solutions are left to the MAClet Decision Logic (synchronization) and MAC program (management of error conditions) defined by the network operator. The synchronization signals can be based on the events and conditions available in the WMP and are specified in the MAClet activation data.

In order to activate a new MAClet on a group of stations, the AP sends a “run” action message to the stations list. If the command does not include an activation data field, each station can start the program asynchronously, i.e. without a common reference signal. If present, the activation data specifies the triggering event that is usually a control frame sent by the AP or the expiration of a (relative or absolute) timer. While the relative timer is in turns expressed as a function of a network synchronization event (e.g. the next channel busy time), for using an absolute time reference the MAClet Control Process has to rely on a time synchronization function. In our infrastructure scenario, such a synchronization is easily provided by the beacon timestamps, while in general scenarios it has to be explicitly supported by the MAClet Distribution Protocol.

Different activation solutions based on a 3-way handshake mechanism can also be defined in the distribution protocol. After the reception of the run message, each station involved in the network reconfiguration sends a confirmation message. When the AP receives all the confirmation messages, it sends an enabling message. Only after the reception of this message, the stations switch to the new MAC program at the occurrence of the next triggering event. Figure 5.2 shows an example of messages exchanged between the AP and two stations for loading two different MAClets (a legacy DCF on station 2 and a TDMA protocol on station 1), whose activation is triggered by the first beacon received after the enabling message.

Chapter 6

MAClet Control Framework

6.1 Introduction

Cognitive and active wireless networks were already promised at the end of the last century [30, 53]. They introduced the idea of dynamically reprogrammed devices in reaction to unpredictable mutating topology, radio contexts and conditions, user or application requirements, spectrum availability. Although more than a decade has passed, this foreseen flexibility does not involve commercial products that are still based on monolithic architectures and which are programmed, once in their lifecycle, with fixed and unmodifiable medium access strategies. The inertia in shifting towards flexible wireless architectures is mainly due to the wrong belief that flexibility means open implementation. Flexibility is generally welcomed by the industry because it introduces benefits both for users and producers, whereas openness is an enemy of the common business model which protects know-how and investments.

This chapter presents MAClet Control framework. This architecture is based on findings presented in [27, 38] and its relation to OpenFlow is presented in figure 6.1. OpenFlow flexibility extends till the LLC layer whereas our MAC-controlled framework covers also the MAC. MAC Control framework, provides a sharp decoupling between Control

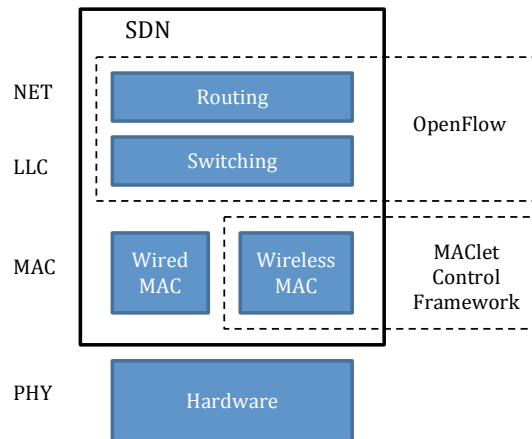


FIGURE 6.1: MAClet Control framework and OpenFlow complementary

plane (i.e. the card behavior) and the data plane (the card capabilities). Decoupling is obtained thanks to two orthogonal elements: the Wireless MAC Processor (WMP)[38] and MAClets [27]. Card capabilities are provided by the manufacturer, they are closed but made available via a defined interface. The MAC processor uses these capabilities as events and actions, the WMP which can be composed to create MAClets. By changing the running MAClets both a single node or a whole wireless network can dynamically change their behavior, even in a multi-vendor scenario.

Running several different MAClets over the same Wireless MAC Processor permits card virtualization (with high-level of separation); executing the same MAClet over multiple WMPs (even from different vendors) permits code mobility and dynamic network reconfigurability.

Let's imagine wireless stations that enter in the coverage of a network able to work with MAClet/WMP enabled AP, as depicted in figure 6.2. The AP is a shared infrastructure among multiple operators, let's say A, B, C . Each station is identified by an index from 1 to 4 and by a label indicating its serving virtual wireless operator¹. Stations are furthermore equipped with chipsets from heterogeneous vendors X, Y . Once stations connect to the AP, they inform it about their operators and the AP (after the AAAs checks) uses well-defined control commands (APIs) to load different MAC behaviors (MAClets) on these wireless cards. MAClet 1, e.g. DCF, on the stations operated by A (sta 2 and 3) and MAClet 2, e.g. pseudo-TDM, on devices operated by B (sta 1).

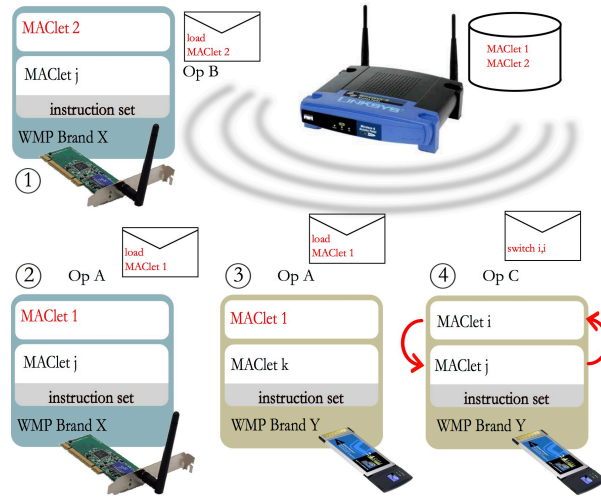


FIGURE 6.2: A simple use case of the MAClet distribution system in a multi-vendor scenario.

The same card loads more (different) MAClets. As for example, cards were running MAClets i and j before the AP sends new MAClets. Previous MAClets are kept in memory in order to be resumed or substituted on a further change of the context². Each MAClet contains a custom MAC protocol and its parameters, both tailored for the specific operator and network context. As shown in figure, heterogeneous devices can run the same MAClet and the same platform can run multiple MAClets.

¹In case a station with an account at the operator D , which is not run in the AP, it can work with standard basic WiFi access

²In the following we will distinguish MAClets stored in the station-level memory storage or in the instruction memory on board the card

6.2 MAClet SDN modeling

SDN is transforming network architecture shifting from today's static networks into flexible, programmable platforms with the intelligence to allocate resources dynamically. This approach permits networks operators and administrators to work on an abstraction of the network rather than of *that* specific network.

An enabling solution for SDN, in the switching and routing environment, is OpenFlow [52]. OpenFlow signs the shift from open source code to open *Application Programming Interfaces (API)*. It is a standard interface designed for SDN which decouples the control plane from the data plane.

This approach comes along with several advantages: simpler and faster programmability, code portability across different vendors' platforms, no need for manufacturers to disclose their internal architecture. Vendors can keep their platform closed continuing protecting their know-how and investments but exposing Open API. OpenFlow permits to modify the behavior of network devices through a remote controller by the mean of a pre-defined *forwarding instruction set* and despite the innovative flexibility introduced by OpenFlow, it presents strong limitations because it is restricted to flow forwarding. Despite SDN concepts are wider than those introduced by OpenFlow, it is going to be *the* SDN enabler and is becoming the predominant SDN standard [8].

However, flexibility provided by OpenFlow is not enough. Although OpenFlow makes network services independent from network interfaces both with wired and wireless networks, its benefits are confined to switching; it limits the impact on wireless reprogrammability. This lack of flexibility emerges also from the analysis of two Open Source control platforms for Software Defined Networks [5, 13] (they consider only routing and switching) and of Pantou [12], an OpenFlow project about wireless networks allows to add a centralised control for legacy tuning knobs such as modulation, channel, MAC address, etc.

Although SDN involves more and more manufacturers of wireless products, it is not clear which opportunities are offered by SDN in wireless scenarios as reported in [33]. On the other hand, wireless access flexibility and adaptability was the driving idea for new wireless access paradigm proposed in [27, 38] with the introduction of the Wireless MAC Processor and the MAClet. Despite the innovating concepts introduced, at the moment of their publishing their similarities and complementarity to OpenFlow were neglected as well as their impact on the SDN paradigm. Surprisingly, although coming from different fields and they use different tools, OpenFlow and MAClet Control framework are impressively similar and complementary.

API openness

Legacy switches implement vendor-dependent flow-tables, OpenFlow identified a common set of functions that runs in many switches and routers.

Although any wireless product manufacturer uses its own implementation of lower MAC/PHY procedures, MAClet Control framework exploits the common set of functions that we named WMP MAC API. It is composed by events, actions, conditions [38].

Device simplification

SDN provides network devices simplification: they do not need to understand anymore several standards because they have only to accept instructions given by the SDN controller.

Legacy wireless cards run inflexible hard-coded MACs, the MAClet Control framework wireless card is simplified because it implements a Wireless MAC Processor that unaware of the MAClet in use, runs it.

Decoupling

Both solutions provide network services decoupled from network interfaces. OpenFlow detaches the control plane from the data plane by decoupling decision on switching/routing and the traffic forwarding.

MAClet Control framework divides the control from the data plane, this is done decoupling the MAC behavior from the low-level resources, as shown in figure 6.3.

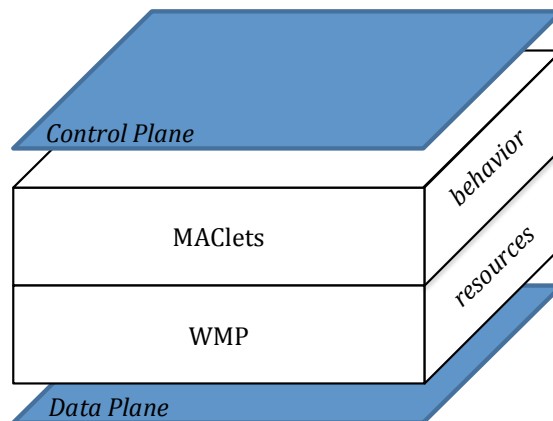


FIGURE 6.3: MAClet Control framework architecture and decoupling between control and data planes

SDN principles make the network as application-customized rather than application-aware and make applications not network-capability-aware rather than network-aware [9]. In MAClet Control framework the same principles are applied: the MAC protocol (MAClet) is unaware about the platform (which has to implement the WMP), the platform is unaware about the running MAClet.

Infrastructure as a Service

OpenFlow provides an open protocol to program the flow-table in switches and routers, MAClet Control framework does the same to program the MAC behavior in wireless nodes. Both approaches permit to consider the network Infrastructure as a Service (IaaS).

Although proposed for the wired scenario, [54] introduced Open Router Proprietary-Hardware Abstraction Layer (Orphal), which uses *switchlets* as modules that have their own address space and thread(s) of control. The switchlets concept remaps in our MAClets, with deep differences on their working principles but with the same goals of portability and mobility.

6.3 Architecture

In this section [28] we present our solution for expressing and implementing high-level network policies in wireless local networks, in contrast to the current solutions based on low-level configurations and vendor-specific implementations. Indeed, many systems require to implement policies that are inherently dynamic and depend on temporal conditions and external events, such as interference measurements, network topology, load conditions, and so on. We present a control architecture for defining these policies and program wireless interfaces to follow them. Our control architecture has the following features:

1. it is based on the WMP API for collecting channel signals and statistics,
2. it exploits frame classifiers for managing multiple virtual interfaces,
3. it adopts meta-state machines for implementing reactive decisions.

Specific control messages allow to configure the desired policy on the nodes and to coordinate the activation of new policies.

6.3.1 Programmable Wireless Nodes and Policies

We assume that our programmable wireless nodes are composed by a WMP enriched with the possibility of defining frame classifiers linked to different MAC programs. Since the MAC Engine is able to switch from a MAC program to another, multithreading can be supported by opportunistically programming the switching events (e.g. at regular timer expirations) in the meta state machine. This feature allows to run simultaneously multiple access schemes over the same hardware (as multiple virtual interfaces with different behaviors). A frame classifier is then required for multiplexing the traffic between the available access schemes. The classifier can work on several frame parameters, such as the QoS class, the source and destination MAC addresses, the frame size, the frame type, the events occurring when processing the frame, etc. On top of the WMP extended platform, a MAC adaptation policy can be programmed by loading a meta state machine and the relevant MAC programs, as shown in figure 6.4(a). The meta state machine can specify a one-time switch from a given program to another, multiple switching events from two or more programs, or even a periodic switch to a doze state program for preventing the node from accessing the medium at regular time intervals. The policy can be transported into MAClets that can (entirely or incrementally) code elementary state machines and code switching conditions. Moreover, it also includes a table mapping the traffic flows into the multiple running programs.

6.3.2 Control System

As in OpenFlow, we envision a system with a clear separation between the *data plane* and the *control plane*. For configuring the data plane, i.e. the MAC programs and the relevant traffic queues at each node, the control plane is responsible for:

1. collecting low-level information for estimating the network context;
2. distributing and configuring MAC programs;
3. ensuring against network inconsistencies in medium access rules.

Figure 6.4(b) shows how the control plane acts on the programmable nodes: different MAC programs are loaded on the nodes and linked to different traffic queues according to the policy programmed by a MAClet Controller. The policy is given by a meta machine describing the switching conditions from a MAC program to another. A MAClet manager is responsible of physically transmitting the relevant programs and loading them on the nodes. MAClet Manager, MAClet Controller and MAClet Repositories are the main components of the control plane. The MAClet Manager handles MAClets at node-level and provides the node-level intelligence. The MAClet Manager transmits and receives MAClet protocol messages to/from MAClet Controllers and Managers. It upgrades the local repository and loads, runs, configures MAC programs over the WMP. The MAClet Controller provides the network-level intelligence on the basis of low-level data received from MAClet Managers; it commits locally computed best response strategies or those decided by the operator. Different controllers can work simultaneously on the same physical network. Finally the MAClet repository stores some basic state machines to be composed into controller policies. A central repository is available for each controller, while a local repository contains the most recent or used MAC programs for a prompt availability at the node level.

6.3.3 Control Messages and Procedures

Policies distribution among wireless nodes is performed in three (cooperating) ways:

1. the controller sends the MAClets to the MAClet Manager of each node via dedicated unicast control messages (specifically acknowledged);
2. the controller sends MAClets in broadcast, by requiring that each MAClet Managers floods them into a whole sub- network;
3. the MAClet Manager of a given node requests the current policy to its neighbors.

In order to avoid policy mismatching among the nodes, it is required to support a distribution protocol for disseminating the policy and a synchronization protocol for coordinating the policy activation. Standard WLAN protocols assume the role of default common protocols to be executed (eventually, on a pre-defined common channel) for supporting a pre-shared communication policy. We assume that the default protocol and configuration parameters are pre-loaded in each WMP with a bios state machine (e.g. in our implementation, the bios machine is a legacy DCF working on channel 1).

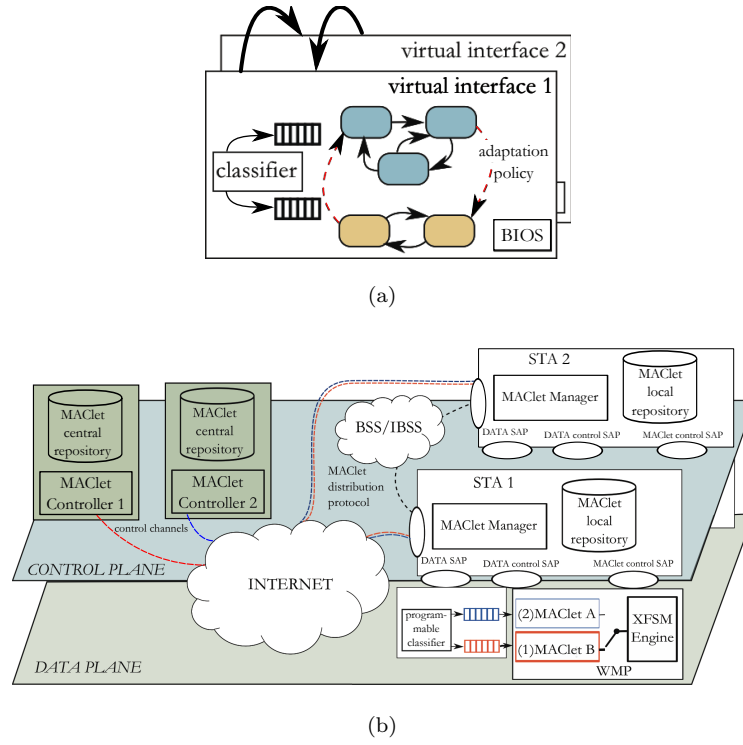


FIGURE 6.4: Architecture detail about Policy Control (a) and the Control and Data Planes (b)

MAClet Management Messages handle control channels Managers / Controllers	
register	(MM to MC). Registration involves one virtual interface
poll	sent by the MC to check if MM are still alive
ack	message acknowledgment
MAClet Action Message Fields	
MAClet	request / send MAClet code
m params	get / set of MAClet parameters
policy	flow control policy
p-params	used to connect queues to MAClets
cmd	send, load, run, dump, set timer
activation_param	triggering event, scheduled time
deactivation_param	triggering event (scheduled time, expiration)
MAClet Information Message Fields collect low-level measures	
type	request: (MC to MM), reply: (MM to MC)
param	freq, collisions, sent frames, ...
Flow Control Messages associate queue to one or more MAClets	
queue_cmd	create, delete, associate
MACletID	identifies the configuring scheduler
sched param	discriminating parameters

TABLE 6.1: MAClet Controller messages

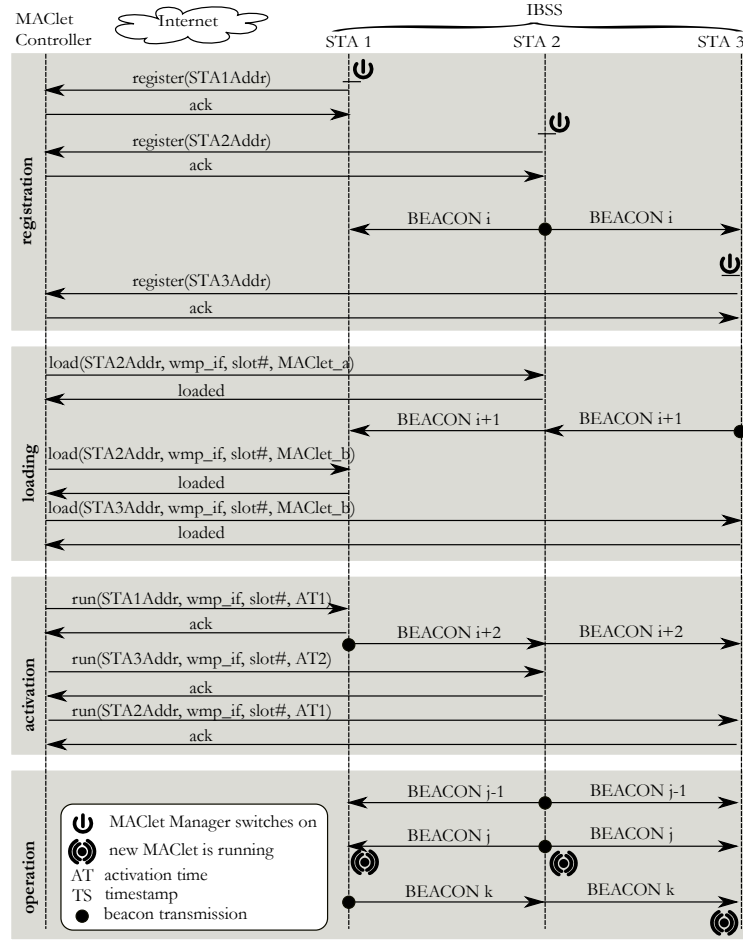


FIGURE 6.5: MAClet Distribution Protocol: message exchange and MAClet synchronization

Control messages are divided in Management, Action, Information, and Flow Control Messages, as summarized in table 6.1. Management Messages allow registration of the MAClet Managers to a given controller. Action Messages are used to send, load, activate, configure MAClets and their parameters, Information Messages carry on low-level statistics from managers to controllers, whereas flow control messages are used by the Controller to create, remove, and configure queues. MAClet Manager registers at one or more MAClet Controllers when switched on; then they periodically refresh their registration. The Manager informs to the Controller about the platform capabilities (e.g. how many MAC programs can be run in parallel, being a platform-dependent parameter). Control procedures are organized into three phases: registration, loading, and activation, as shown in figure 6.5. These phases can partially overlap each other (stations asynchronously register) or be merged (a single message can carry the loading and the activation commands). The registration phase creates or refreshes a virtual interface on the card and delegates its control to a MAClet Controller. The access policies for the newly created virtual interface are then provided and started by the MAClet Controller by means of the loading and activation commands. The Control Plane also provides the primitives for programming the desired synchronization and error recovery operations. Network-level policies require coordination among nodes. For example, the activation of a new program may require a common reference signal for avoiding critical

inconsistencies (such a temporary use of different transmitting channels, mismatch in slot assignment, etc.) leading to disassociations or other network errors. A new policy can be executed upon reception (if the command does not specify an activation data), or at the occurrence of a the triggering event (such as a control frame sent by the AP or the expiration of a (relative or absolute) timer. While the relative timer is expressed as a function of a network synchronization event (e.g. the next channel busy time), for using an absolute time reference the controller has to rely on a time synchronization function. Different activation solutions based on a 3-way handshake mechanism can also be defined in the distribution protocol.

6.3.4 MAClet Manager

This is the local software executed into programmable nodes and interacts between the MAClet received and the WMP. MAClet Manager extract the bytecode from the MAClet message and define a set of load/inject/execute commands. bytecode injection can be executed locally by the host where user is able arbitrary to inject a bytecode without using MAClet but it main functions is to receive and handle MAClet message. The current version is currently an extension of “bytecode-manager” v2.47 and it can be obtained on WMP git repository ³. MAClet Manager is written in C and it is compatible with x86 and OpenWRT system, it use a C library called wmlib.h that give a framework for extend internal communication between Userspace and NIC driver. The Userspace/NIC interaction is made with debugfs interface of b43 driver used to establish interaction between userspace and driver. The current version of MAClet Manager is able to communicate with a specific broadcom vendor card, BCM43xx, in [B](#) a brief technical description of the chipset. Moreover wmlib is designed to extend compatibility with other kind of driver interfaces, such as nl80211⁴. In figure 6.6 a stack description of wmlib for a generic customized application.

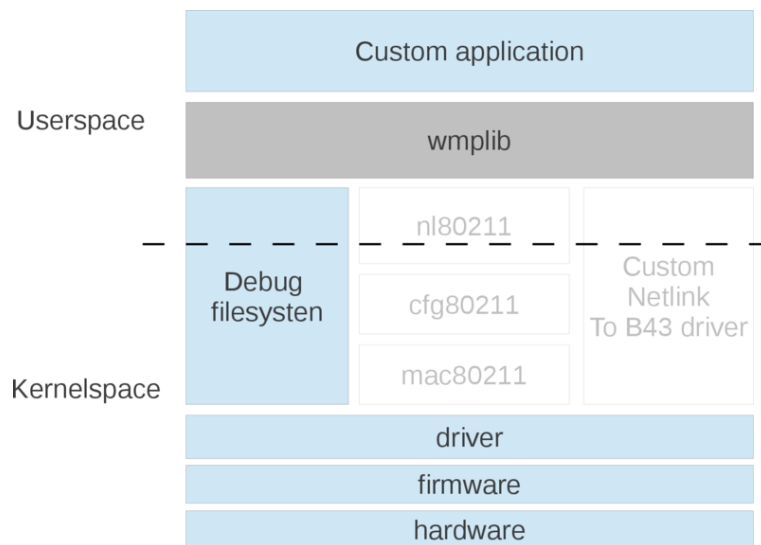


FIGURE 6.6: wmlib for MAClet/WMP interaction

MAClet Manager has been developed as a client/server application. The server stations listen for MAClet packets and run the command or load the bytecode. This tool is to

³<https://github.com/ict-flavia/Wireless-MAC-Processor>

⁴<http://wireless.kernel.org/en/developers/Documentation/nl80211>

be considered as a daemon that run continuously, take sensing information from the Node status, receive the MAClet messages and interact with WMP. MAClet Manager command description is Appendix A. Sensing information can be collected and sent to the MAClet Controller which analyze the network status and reacts sending an action MAClet message with a new MAClet, or an activation message or a parameter list update message.

6.3.5 MAClet Manager as a Controller

In very simple scenarios, when a node must send MAC programs to other stations, MAClet Manager has been used as controller. MAClet Manager operating in client mode defines a set commands to execute in remote run/load/switch of bytecode. A control script written in some language (i.e. bash, python or perl) governs distributed commands based on the controller station, for example in Access Point.

6.4 OMF for MAClet Controller

OMF^[6] (ORBIT Measurement Framework) is a control, measurement and management framework for experimental platforms. From an experimenter's perspective OMF provides:

- a domain-specific language, named OEDL, which allows the user to:
 - describe the resources needed for an experiment, and their required configuration
 - describe the applications to use in an experiment, and the measurements to collect from them
 - describe the different tasks to perform during an experiment, and the time or event that trigger them
- a set of software tools, which:
 - accept as an input the previously described experiment
 - initialize and configure the needed resources with the required configuration and applications
 - send commands to the resources to effectively execute the described experiment tasks at the appropriate time/events
 - collect measurement data from the applications and/or the resources
 - access and analyze the resulting experimental data

OMF⁵ has an instrument tool called OML (ORBIT measurement library) that allows application writers to define a set of measurement points. While an experiment run and executes an OML-ized application, OMF framework can access to the measurement points and store it on database. OML is quite flexible and it can be used to collect data from any sources, this feature is very interesting for dynamic adaptation usecases

⁵https://mytestbed.net/projects/omf/wiki/An_Introduction_to_OMF

and for the Sensing requirement of MAC Control Plane. OML consists of two main components:

- OML Client library: is a C API to collect the applications measurement. The library includes a dynamically configurable filtering mechanism that can perform some processing on each measurement stream before it is forwarded to the OML Server.
- OML Server is responsible for collecting and storing measurements inside a database.

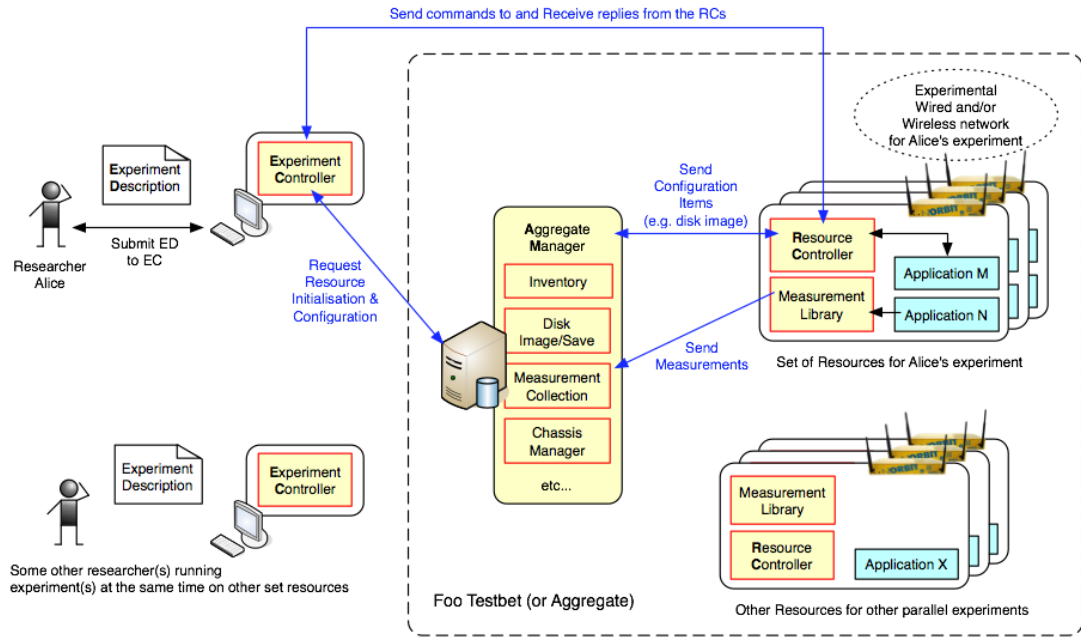


FIGURE 6.7: OMF System Architecture

The Architecture of OMF is shown in fig. 6.7. Experiment Controller (EC) is the node that interprets the Experiment Description (ED). EC send the configurations and the resources request to the Aggregate Manager (AM); AM also store the Measurements. When the experiment nodes are configured EC send the command to the Resource Controllers (RC) which run on each node; At this stage the experiment is running and Application may collect measurement and send it to Aggregate Manager.

Chapter 7

Experiments and Evaluations

7.1 Introduction

In the following sections have been described the usecases of interest as a proof of concept of the WMP/MAClet SDN architecture; These experiments shows network example usecases for exploit reconfigurable and reprogrammable MAC. Virtualization, MAC program opportunistic injection and physical sensing optimization are three typical scenarios addressed in our usecases. The main purpose is to describe how the existence of a centralized control node is a programmable network solution in mobile dynamic environment.

7.2 Virtualization

7.2.1 Scenario Description

[34] In this use case we assume that the same Access Point (belonging to a public network) is shared between two different WiFi operators. The scenario is obviously not new, and indeed it has been specifically addressed by many manufacturers that allow to define Virtual APs, each advertising a distinct SSID and capability set. Virtual APs allow operators to share the same physical infrastructure, while offering access to distinct networks, but they typically suffer of a scarce level of *isolation*, since the resources allocated to each one cannot be really partitioned when stations employ random access schemes and suffer of unpredictable interference. Suppose that the two operators want to implement a different service model: the first operator (operator A) advertises “FIXED” SSID, offering access to the Internet with a fixed (guaranteed) bandwidth, while the second one (operator B) advertises ”BEST” SSID, offering a traditional best effort access. Although the standard includes PCF and HCCA for managing the medium access by means of polling, the lack of support in commercial products prevents an easy solution. Using MAClets, the resource repartition between the two operators can be addressed in a very effective and flexible manner. If all the stations employ a MAClet Control architecture, each operator can send a MAClet to the associated stations for enabling the medium access at regular time intervals (for example, in a fraction of the beacon interval reserved to the specific operator) and preventing it in the rest of the time. Moreover,

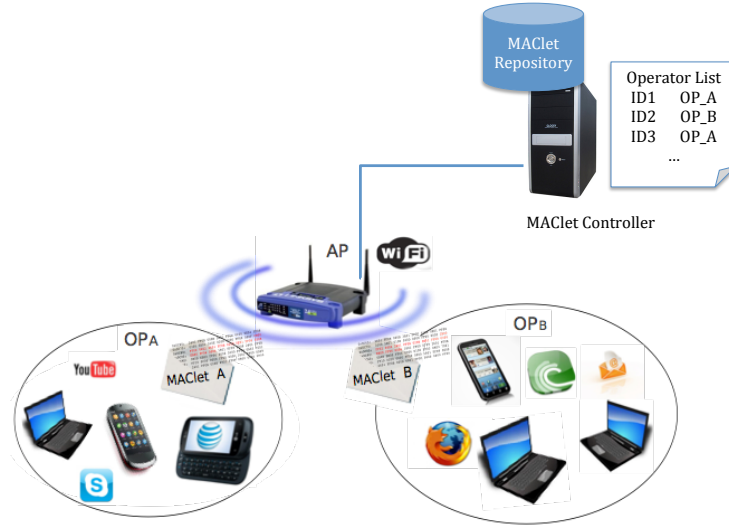


FIGURE 7.1: two operators share the same physical Access Point by providing two different access services to their clients (fixed-rate and best-effort).

the time reserved to each operator can be dynamically tuned (by updating the MAClet configuration parameters) according to the traffic conditions and to the agreements between operators. Multi-Operator MAClet are stored on MAClet Controller. Each node start with a default MAC program that implements a DCF standard. After a legacy 802.11 Association, AP send an information message to Controller with the new node attached informations, typical MAC address is used for identify the node. At this point the MAC controller component designed for MultiOperator check the node in an operator list and decide the MAClet to send. In this scenario Controller communicate to AP and sends the MAClet for the new node. At this point AP send the MAClet and inject the bytecode for selected operator that will work on the selected Operator MAC.

7.2.2 MAC Virtualization and Synchronization

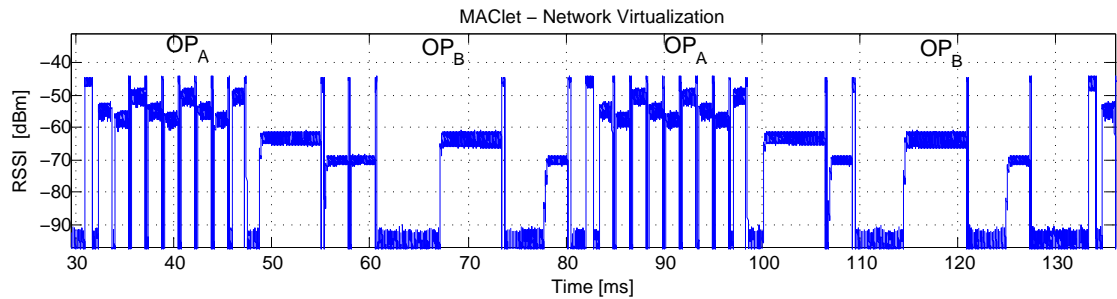


FIGURE 7.2: An experimental trace of network virtualization: operator A and operator B use the channel in different time intervals with independent access schemes (TDM and DCF).

Different solutions are possible for addressing the beaconing and the MAC pausing in the above scenario. We chose to transmit two SSID Information Elements within each beacon, thus leaving the beacon interval unchanged. The MAC pausing has been programmed in a meta machine between the operator-dependent MAC program and a simple state machine with a waiting state only. At the expiration of the pausing time, each station enters the waiting state until a new activation event is revealed. In the waiting state, the stations continue to receive beacons from the AP for keeping the synchronization to the time interval of their operator.

According to the SSID specified in the association request, each station receives a different MAClet: a legacy DCF program for the stations associated “BEST” SSID, and a TDMA program for the stations associated to the “FIXED” SSID. The DCF MAClet is a legacy DCF protocol that is suspended at the reception of a new beacon. The reactivation is triggered at the expiration of a parametric timer set before the suspension. The opposite activation and deactivation actions are performed for the TDMA MAClet. This mechanism guarantees a perfect coexistence and isolation between the two networks, since stations accessing the medium during the same time interval employ uniform channel rules, and no station associated to a given operator can interfere with the other operator network. Isolation is not obviously guaranteed with other external interfering networks.

The configuration parameters of the DCF MAClet are the DCF contention parameters that are uniformly set to all the stations (although some forms of user prioritization could be easily supported by differentiating these parameters in the MAClet directed to each station). Conversely, the MAClet transmitted to a new station associated to the “FIXED” SSID specifies a different program parameter indicating the slot numbers allocated to the station (multiple slots can be allocated to the same station). In each TDMA slot, frame transmissions still follow a 2-way handshake mechanism. When the MAClets are loaded on a new arriving station, the reception of the first beacon frame activates the execution of the program. Subsequent beacons are used as synchronization events for pausing the DCF programs and resuming the TDMA ones, as well as for activating the DCF suspension timer and computing the beginning of the TDMA slots. Although beacon frames are scheduled at regular time intervals, they can be delayed because of ongoing frame transmissions. These transmissions can be due to external interference, but also to stations associated to the “BEST” SSID starting a frame transmission right before the expiration of the operator time (no control is indeed implemented on the residual time before starting a transmission). In case of delay, to guarantee the fixed rate of TDMA stations, the time allocated to the “BEST” SSID operator can be reduced in the subsequent beacon interval. The possibility to dynamically tune the DCF activation time can also be exploited for performing a dynamic repartition of the resources allocated to each operator.

Figure 7.2 shows an example of resource repartitions between operators A and B in two consecutive beacon intervals. The figure plots the channel activity trace captured by the USRP: for better distinguishing the two virtual networks, the TDMA stations transmit at 11 Mbps while the best effort stations transmit at lower data rates (5.5 Mbps and 2 Mbps). Note that in the first TDMA slot the channel is busy (i.e. a transmission has been originated in that slot), but no acknowledgment is received because of channel errors.

7.2.3 Performance Evaluation

We setup a testbed with a fixed number of stations associated to the “FIXED” SSID and a time-varying number of stations associated to the “BEST” SSID. Specifically, three stations access the channel by using TDMA, while five stations join sequentially the best-effort network at regular intervals of one minute. The TDMA frame is organized in nine allocated slots, uniformly assigned to all the stations (three slots each). The beacon interval is set to 50ms, while the slot size is set to 1.7ms (enough to accommodate the

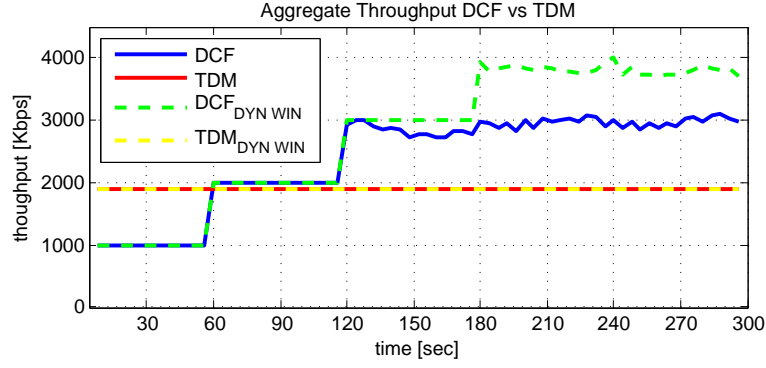


FIGURE 7.3: Resource repartition between two different operators using different access rules (TDM and DCF).

transmission of a payload equal to 1470 byte at 11 Mbps). All the stations transmit at 11 Mbps.

We repeated two different virtualization tests: in the first one, each operator receives an equal share of the available bandwidth (i.e. the activation time is one half of the beacon interval), while in the second one, the TDMA operator agrees to release the available bandwidth to the other operator. TDMA stations have a traffic rate of 630 kbps (smaller than the maximum guaranteed bandwidth, namely $3 \cdot 1470 \cdot 8/50ms = 705.6kbps$), in order to have a non-null probability to have some slots empty. DCF stations work with a traffic rate of 1 Mbps.

Figure 7.3 shows the per-operator throughput results obtained in both the experiments. In case of equal share of the bandwidth, after the third station joins the network the throughput of the best-effort operator (blue curve) saturates to about 3 Mbps (i.e. one half of the total network capacity at 11 Mbps). TDMA network is obviously under utilized because it consumes only 1.89 Mbps (being 3 Mbps the available capacity). By adjusting the time allocated to the best-effort operator, the third station can join the network without causing any throughput degradation. The aggregated network throughput (green line) for the best-effort network is now about 4 Mbps, while TDMA stations performance are not affected by increased DCF traffic.

7.3 Cognitive Network MAClet switching

In this testbed we exploit WMP, MAClets and the control architecture envisioned in [36]. Here the WMP is a MAC-agnostic wireless engine that executes MAClets, namely state machines that, together with an initial state and an activation event, encode the actual MAC programs. The MAClet Controller provides the network-level intelligence, i.e., it makes decisions according to medium access parameters and spectral usages perceived by nodes, and sends MAClets and MAC/PHY configuration data to MAClet Managers running on every network node, so that the channel access strategy can be dynamically adapted to network conditions. The cognitive paradigm is implemented in this testbed thanks to multiple levels of MAC adaptability. MAC policies can be dynamically changed by the MAClet Controller through total or incremental updates. The adaptation policy is a dynamic switch between MAClets and results in high-speed node-level cognition based on node-local awareness. The meta-machine that defines the adaptation policy is a hierarchical XFSM composed by two or more XFSMs and is (again) a MAClet.

As decisions depend on local awareness and measurements, events and conditions, this meta-machine implements the *node-local MAC adaptation policy*. The *cognitive global loop* provides low-level info to the MAClet Controller which makes decisions for *network-level adaptability* following a network-wide awareness approach. As reactions driven by the MAClet Controller are slowed down by communication delays, network latencies, MAClet injection and activation, the duration of a cognitive global loop can sum up to several seconds. In figure 7.4 The summary description of the loop logic. Three elements compose the cognitive paradigm: *measurement*, *decision*, *enforcement*. MAClet controller keeps the state of the network with measurement of physical statistics, and take a decision based on the results of the values of the measurement set. At this point MAClet are sent to the stations of the network to enable a specific state machine and execute the cognitive enforcement phase. In figure 7.5 a performance evaluation

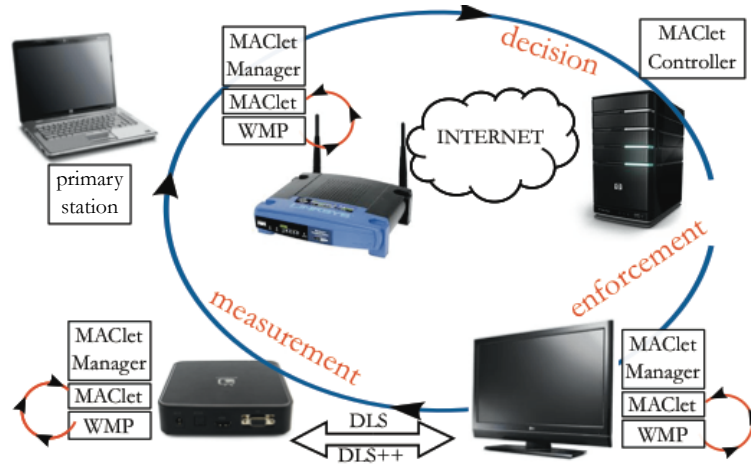


FIGURE 7.4: Cognitive Loop DirectLink

of the Cognitive loop implementation. The measurement is taken counting the number of collision on AP, when the number of collision rise above a given threshold MAClet Controller send a MAClet to WMP stations. To obtain a performance improvement MAClet controller send a Direct Link Setup (DLS) MAClet. DLS MAClet contain a MAC program with the implementation of a Direct Communication MAC protocol between two assigned stations. This Mechanism is not nearly new, and indeed was specifically addressed by the 802.11e task group with the introduction of the Direct Link Setup (DLS), further extended in the 802.11z-2010 amendment. However, a direct link setup is not automatic (i.e. the kids should take care of changing the settings of the TVset during the streaming!). Moreover, the direct link uses the same wireless channel, thus, although to a lower extent, the station connected to the Internet still suffers of a bandwidth reduction. Another enhanced implementation of DLS is the so called DLS++, a version of DLS where the Direct link stations assign a specific channel different from the BSS channel. with DLS++ is possible to isolate the directlink stream acting offloading for the legacy network. DLS and DLS++ XFSM implementation acts a per-frame direct-link forwarding reformatting the of MAC source/destination address in MAC layer using WMP action specified for MAC header modification. Furthermore in DLS++ MAC program the Channel selection is provided introducing a periodically jump to the main channel to allow nodes to maintain the association to AP, in this way both Directlink and non-Directlink flow can be managed.

In the experiment, DLS activation causes the immediate improvement of the DirectLink stations performance. Another decision step can be taken if the number of collisions

still increases. MAClet Controller send a new MAClet with a DLS++ an it is injected into WMP node giving a complete offload of the primary channel.

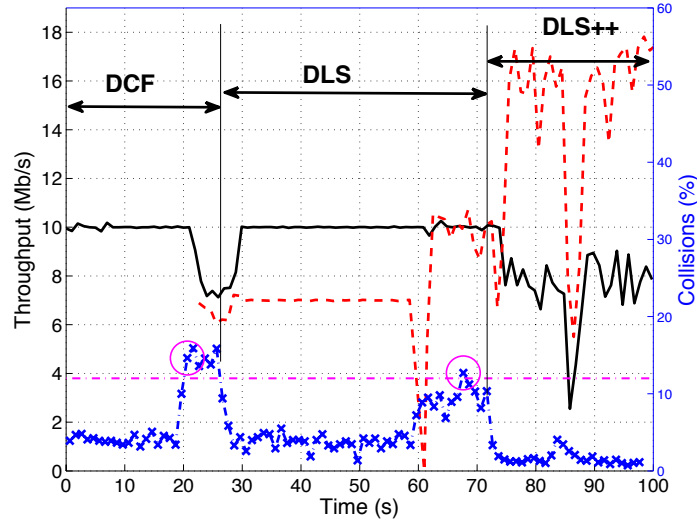


FIGURE 7.5: MAClet Activations

7.4 Cognitive Channel Selection

Another interesting experiment of a cognitive usecase is the automatic channel selection. The logic of this Cognitive Channel selection implementation is to define a sensing period where nodes collect busy time informations and a period where all nodes use the optimal channel chosen by MAClet Controller. In Ad-hoc Network nodes provide a periodical busy/idle estimation on channels and send sensing information to MAClet Controller, at this point on MAClet controller a decision component choose the best channel and send a MAClet Parameter message to configure network nodes. A way to address communication between wireless nod and MAClet Controller is to introduce a wireless interface on it in this way MAClet Controller is albe to send and receive MAClet messages; other solution could be to use a wireless gateway node. For this use case has been used a gateway node wired connected to MAClet Controller. WMP MAC program implements a DCF standard with enhanced internal policies: a automatic channel switching during busy channel estimation period and a special frame dissection analysis. During channel busy estimation phase, sender deliver frames with enhanced information header and store the busy channel information, repeating this operation for a defined number of channels, 3 in this experiment. Enhanced channel information allows receiver to change dynamically its channel during the busy estimation avoiding bandwidth waste.

When the channel busy estimation phase is concluded nodes send to gateway node the sensing measurements, MAClet Controller analyze the sensing measurement for each node and choose the channel with the lower average busy time. At this point MAClet Controller sends a MAClet to the gateway node containing the channel parameter. MAClet is shared throw the wireless nodes and new parameter channel is injected, this complete the enforcement MAClet distribution.

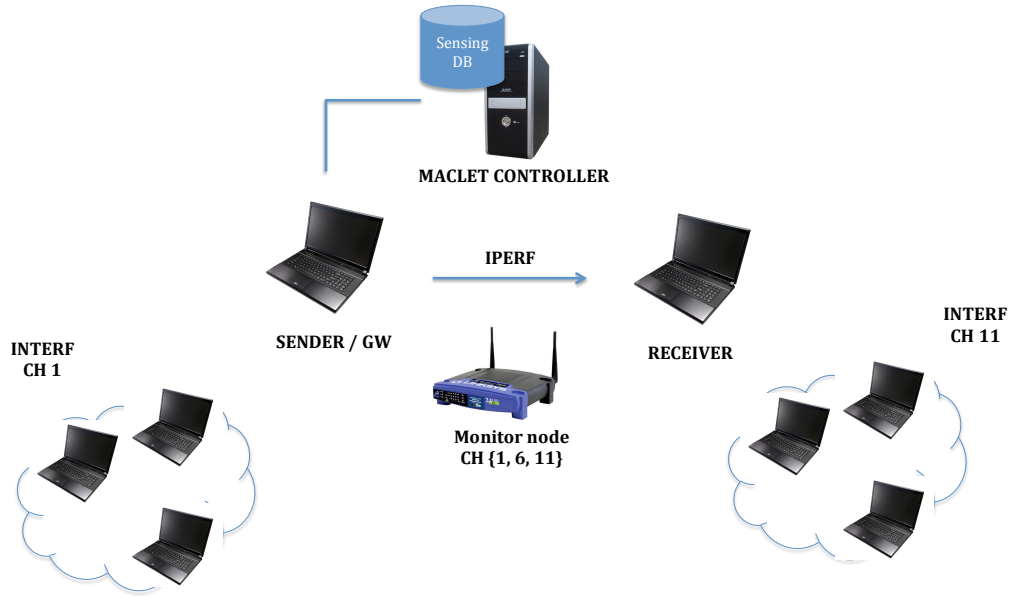


FIGURE 7.6: Channel hopping experiment setup

7.4.1 Experiment setup and evaluation

As described in figure 7.6, in this experiment two WMP nodes associated in ad-hoc mode run an iperf session, modulation capacity is 6Mbps and the traffic is in saturation. A monitor node captures the traffic on the channels to intercept the channel traffic. As a proof of the effective channel selection the experiments have been conducted using two channels with intensive channel usage and one channel without co-channel interference. Figure 7.7 shows the Throughput fluctuation during the channel busy estimation and the channel choosing at the end of estimation. The experiment proof has been designed with 3second per each channel estimation repeated each 20 second. Channel estimation and repeat estimation period can be tuned defining the time parameters of internal policies of MAC program.

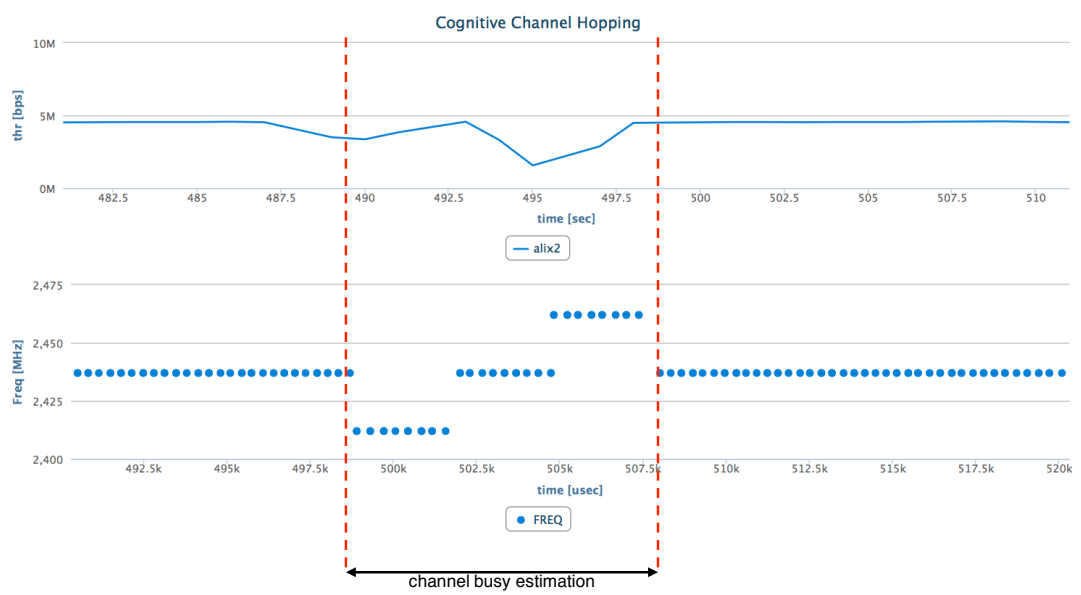


FIGURE 7.7: Channel hopping evaluation

Chapter 8

Experimental Contribution for vehicular environment networks

8.1 Introduction

In this thesis we discussed about the impact of applications and networking scenarios on the performance of WLAN networks, and on the potential benefits of simple adaptations (especially at the MAC layer) for dealing with scenario peculiarities. Indeed, we have shown that MAC layer flexibility can significantly improve the network performance if a few extensions or parameters tunings are introduced as a function of the so called network context.

A special application scenario considered in this work (whose relevance is becoming more and more important for future cities) is represented by vehicular networks, i.e. networks made of cars and infrastructure units disseminated along the roads (called roadside units) that can be primarily used for safety applications (but also for disseminating different type of information data). These networks (also called VANETs) are characterized by very short contact times between the network nodes (either in the case of vehicle to vehicle V2V communications, or in the case of vehicle to infrastructure V2I communications) and by potential inefficiencies of the physical layer. To cope with this specific scenario, different 802.11 standard extensions have been proposed, mainly for shortening the time required for creating the wireless links and for allocating special non-interfered bands for safety applications. However, the same adaptations could have been performed to programmable wireless interfaces (if commercially available).

In this chapter, we describe some experimental results carried out at the University of California Los Angeles (UCLA) for comparing the performance of these emerging standards with the performance of old standards (namely, 802.11a and 802.11g), whose operating conditions have been configured for emulating the novel standard functionalities. Results show that the most discriminating factor is represented by the different interference conditions experienced in the ISM bands. Overall, 802.11a performance can be comparable with 802.11p performance, being the 802.11a bands much less interfered.

8.2 VANET standards

The protocol stack for VANETs based on IEEE standards is defined by the IEEE 802.11p and IEEE 1609.x (WAVE) family of standards. IEEE 802.11p is an amendment of the IEEE standard 802.11, specifying extensions to 802.11a, adapting it to DSRC communications in the 5.9 GHz band. A description of the modulation techniques allowed for those communications, the operations that must be performed by each layer, interfaces and primitives used to ensure communication between different layers, and other important information are described by the IEEE 802.11 standard. IEEE 802.11p provides the extensions to fit the VANET requirements, that result from the high mobility and the need to exchange messages between nodes without association (communicating outside the context of a Basic Service Set (BSS)). WAVE does not use authentication and association to decrease communication latency. Some parameters specific of the IEEE 802.11p standard are the transmit power limits, channel spacing and frequencies. As a normative standard, the main role of the IEEE 802.11p amendment is to define the minimum set of specifications required to ensure interoperability between wireless devices, produced by different manufacturers. The specifications include:

- Functions and services required by stations to operate correctly and to exchange messages without joining a BSS;
- Signaling techniques and interface functions used by stations to communicate outside the context of a BSS.

[22] On the other hand, the IEEE 1609.x family of standards for WAVE addresses the homogeneous communications interfaces between different manufacturers and provides a foundation for the organization of management functions and modes of operation of system devices. The WAVE standards define the architecture (complementary to 802.11p), a standardized set of services and the interfaces that collectively enable secure V2V and V2I/I2V wireless communications. Together, they provide the foundation for a variety of applications such as those mentioned in Section I. The IEEE 1609.x consists of the following five standards (see figure 8.2):

- IEEE P1609.0 - Architecture - describes the WAVE architecture and services for multi-channel DSRC/WAVE devices to communicate in a mobile vehicular environment (still a draft standard).
- IEEE 1609.1 - Resource Manager - specifies the services and interfaces of the WAVE Resource Manager Application. It describes the data and management services offered within the WAVE architecture, while also defining command message formats and the appropriate responses to those messages, data storage formats that must be used by applications to communicate between architecture components, and status and request message formats.
- IEEE 1609.2 - Security Services for Applications and Management Messages - defines secure message formats and processing. This standard also establishes the circumstances for using secure message exchanges and how those messages should be processed based upon the purpose of the exchange.

- IEEE 1609.3 - Networking Services - defines network and transport layer services, including addressing and routing, to support secure WAVE data exchange. It also defines Wave Short Messages (WSM), providing an efficient WAVE-specific alternative to Internet Protocol version 6 (IPv6) that can be directly supported by applications.
- IEEE 1609.4 - Multi-Channel Operations - provides enhancements to IEEE 802.11 MAC to support WAVE operations, namely number and type of channels, channel routing and coordination, QoS mechanisms and node synchronization.

8.2.1 802.11p/WAVE

In Figure 8.2 an overview description of DSRC stack. IEEE 802.11p WAVE is only a part of a group of standards related to all layers of protocols for DSRC- based operations. The IEEE 802.11p standard is limited by the scope of IEEE 802.11, which is strictly a MAC and PHY level standard that is meant to work within a single logical channel. All knowledge and complexities related to the DSRC channel plan and operational concept are taken care of by the upper layer IEEE 1609 standards. In particular, the IEEE 1609.3 standard covers the WAVE connection setup and management. The IEEE 1609.4 standard sits right on top of the IEEE 802.11p and enables operation of upper layers across multiple channels, without requiring knowledge of PHY parameters.

8.2.1.1 802.11p - Physical Level

The principal Physical characteristic of 802.11p is to divide the spectrum in 7 channels of 10MHz as shown in figure 8.1. Channel 178 is defined as Control Channel (CCH), restricted for safety communications, channel 172 is defined as High Availability and Low Latency (HALL), others are service channels (SCH).

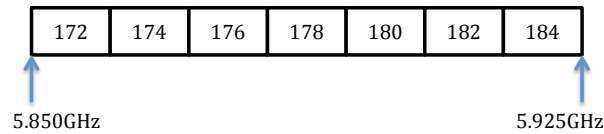


FIGURE 8.1: 802.11p channels

8.2.1.2 802.11p - MAC Layer

The fundamental access method of the IEEE 802.11 MAC is a DCF known as carrier sense multiple access with collision avoidance (CSMA/CA). The DCF shall be implemented in all STAs. The IEEE 802.11p MAC layer is equal to the IEEE 802.11e Enhanced Distributed Channel Access (EDCA) Quality of Service (QoS) extension [12]. This scheme is similar to the standard IEEE 802.11 CSMA/CA scheme called distributed coordination function (DCF), but EDCA can differentiate between 4 different application categories (AC): background traffic (BK), best effort traffic (BE), voice traffic (VO) and video traffic (VI). Different Contention Window (CW) and Arbitration Inter Frame Space (AIFS) values are chosen for the different application categories, where VI has the

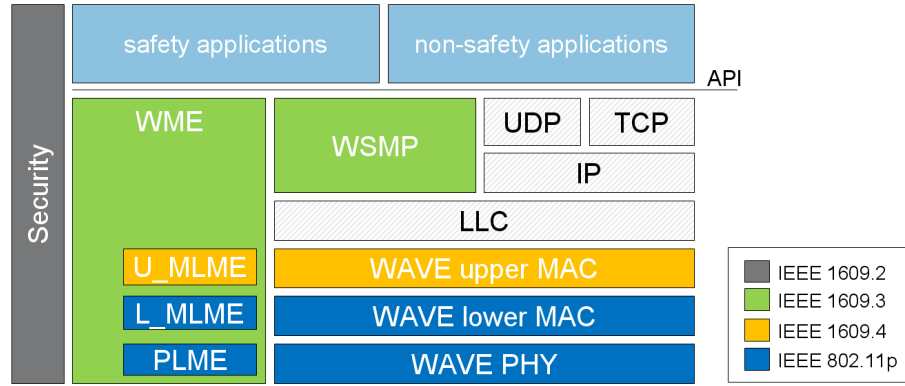


FIGURE 8.2: DSRC STACK

highest priority and BK the lowest. The values used by IEEE 802.11p where specifically chosen for vehicular communication scenarios. Other specific parameter values also distinguish the MAC layer of the IEEE 802.11e and IEEE 802.11p standard: in the latter the SIFS interval is 32 us and the slot time is 13 us.

8.2.2 Motivation

The first motivation about the comparison between 802.11p and legacy 802.11a/g standards is to understand if current standards can provide the performance promised in 802.11p. Obviously a dedicated frequency bandwidth is an important reason to prefer 802.11p, but in case of high density vehicle scenario when CCH is often busy, we can expect that a legacy technology can provide similar performance. Moreover, the newest commodity cards support by default 10MHz bandwidth, in particular 802.11a allows, as a standard, the use of 10MHz bandwidth. Regards the low-latency association time, this requirement can be easily addressed with a smart designing of attach techniques or by defining a communication system able to work neglecting association. Current mac80211 [4] implementation in GNU/Linux Systems, for example, provide the so called “packet injection” functionalities where stations, configured in monitor mode, can transmit custom frames by-passing the association rules implemented by mac80211 modules and injecting the frame directly in the driver queues.

This measurement campaign wants to discover the performance differences between the emerging 802.11p standard and legacy IEEE 802.11.

8.2.3 Metodologies

To obtain a fair comparison between IEEE 802.11p and IEEE 802.11a/g has been decided to define an experiment without an association functionalities, in this way we can design experiment focused only on MAC and physical characterization.

experiment setup

The experiment is designed as follow: two cars are equipped with 3 embedded stations; each node is equipped with specific standard IEEE 802.11 NIC. On car roof are installed

three omnidirectional antennas and a software control tool is used to launch simultaneous experiments. To ensure a fair comparison between cards, a wired bench comparison test has also been conducted to proof that all the nodes transmit with the same power level and receive with the same sensitivity.

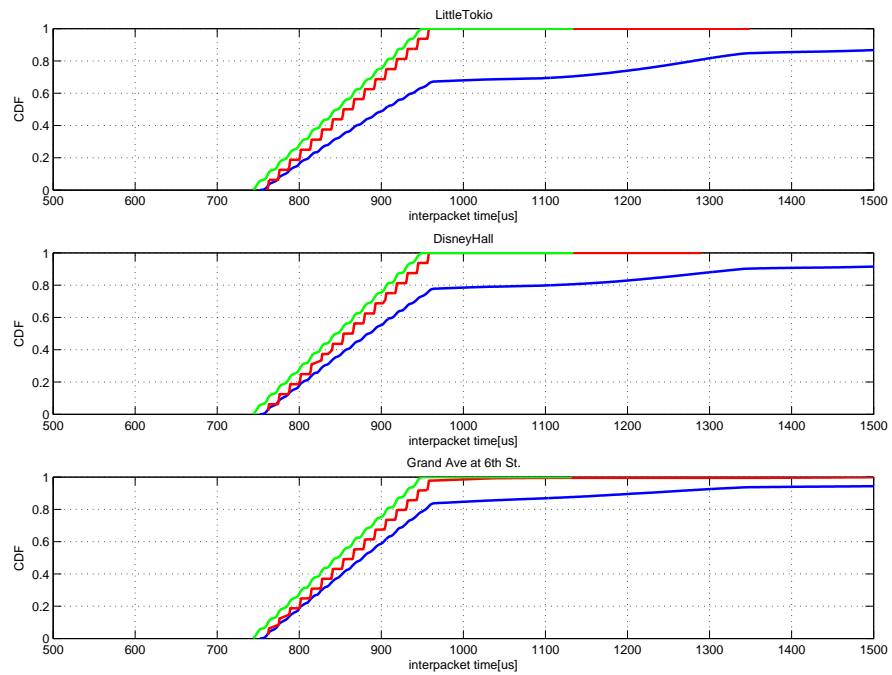
The transmitter generates broadcast raw traffic in saturation with fixed frame size of 200B, fixed transmission power and fixed modulation scheme at 6Mbps. To ensure a fair comparison between 802.11p and other standards all nodes transmit using a bandwidth of 10MHz. Experiments have been conducted in two main urban environment: downtown, residential. We analyzed three different experiments in downtown environment and three different experiments in residential environment. While in residential environment is not expected high interference and multipath, in downtown environment, co-channel interference and multipath highlight an hostile environment that can compromise the traffic performance. During the tests, receiver car is stationary transmitter instead moves following predefined road path, while inside three nodes do simultaneous transmissions. The frequencies adopted for the experiment are: 11g at 2.412GHz, 11a at 5.745GHz and 11p at 5.890GHz.

We compare the distribution of inter-arrival time for each technology and measure the amount of coverage for 99% 70% 50% of average packet delivery ratio. The interarrival time shows the maximum transmission delays and coverage threshold comparison is interesting to understand the safety application requirement that a technology can satisfy.

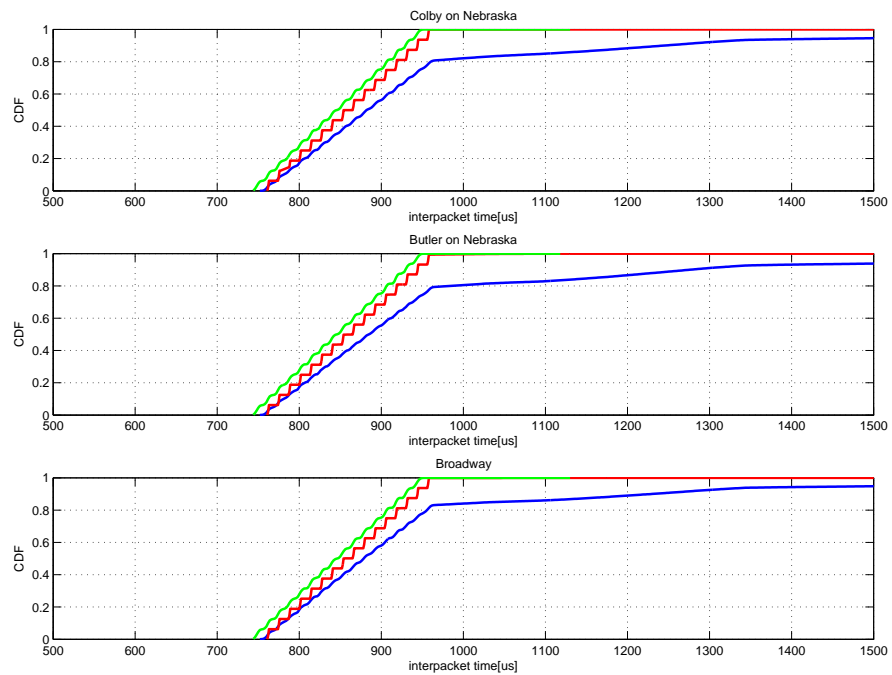
8.2.4 results evaluation

In figure 8.3(a) 8.3(b) interarrival time CDF comparison between three technologies for three experiments. In both downtown and residential environments, 802.11g suffers of transmission delay due to co-channel interference while 802.11a and 802.11p have the same behavior, considering that 802.11a uses the legacy IFS and 11p uses its standard specific IFS. Also for coverage analysis, 802.11g cannot satisfy a coverage requirements for safety applications. Instead, 11a and 11p have a comparable behavior. Moreover the coverage experiment shows better results in residential areas for 11a, despite the downtown experiments where 11p has the best behavior. These results have also been compared with channel utilization measurement during the experiment. The results in figure 8.5 show the channel busy time during the test. In absence of transmissions the busy time for 802.11a is null, while for 802.11g it is not null. This is the proof of diffuse co-channel interference in 2.4GHz.

In conclusion this experiment campaign proofs that current legacy 802.11a standard can provides the same medium access behavior of 802.11p and that the environment plays a very important role for MAC behavior characterization. Moreover, the differences of IFS do not show significant improvements.

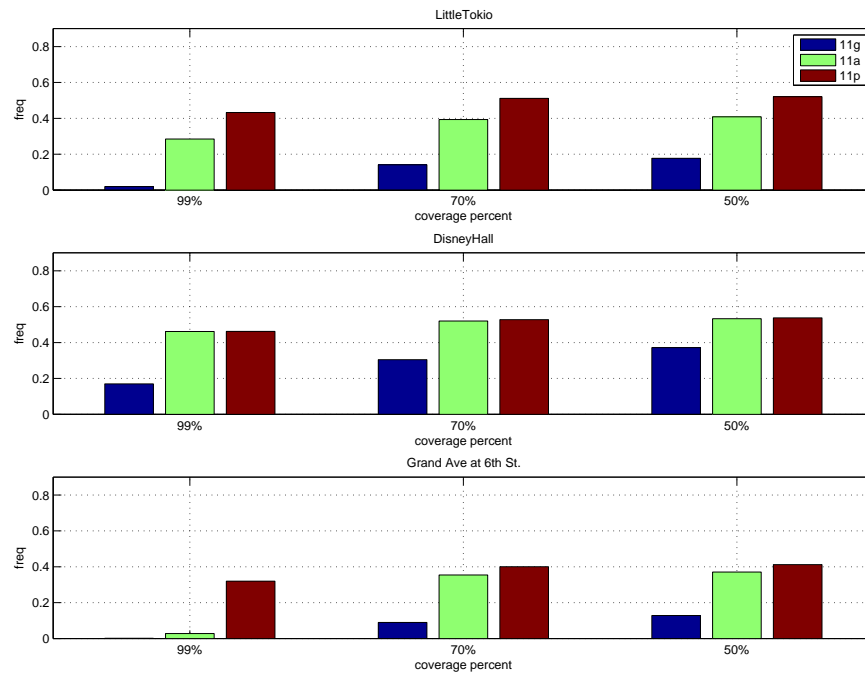


(a) Downtown

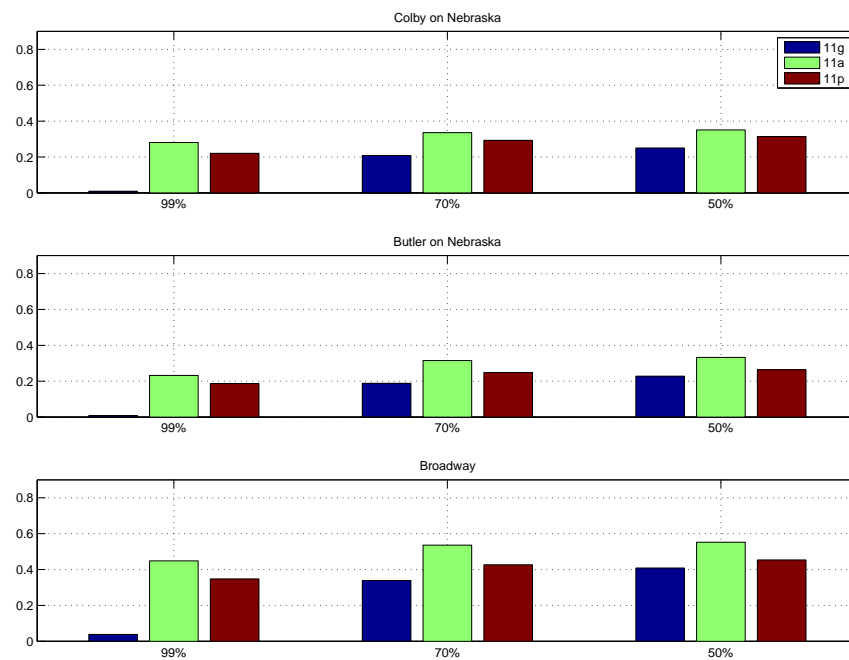


(b) Residential

FIGURE 8.3: Interarrival time - 6Mbps - 10MHz

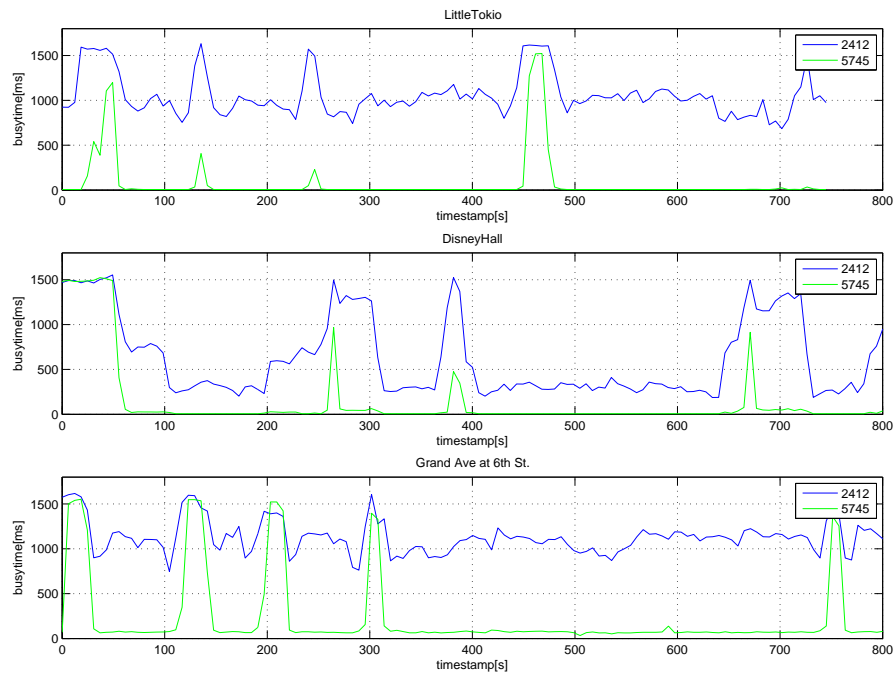


(a) Downtown

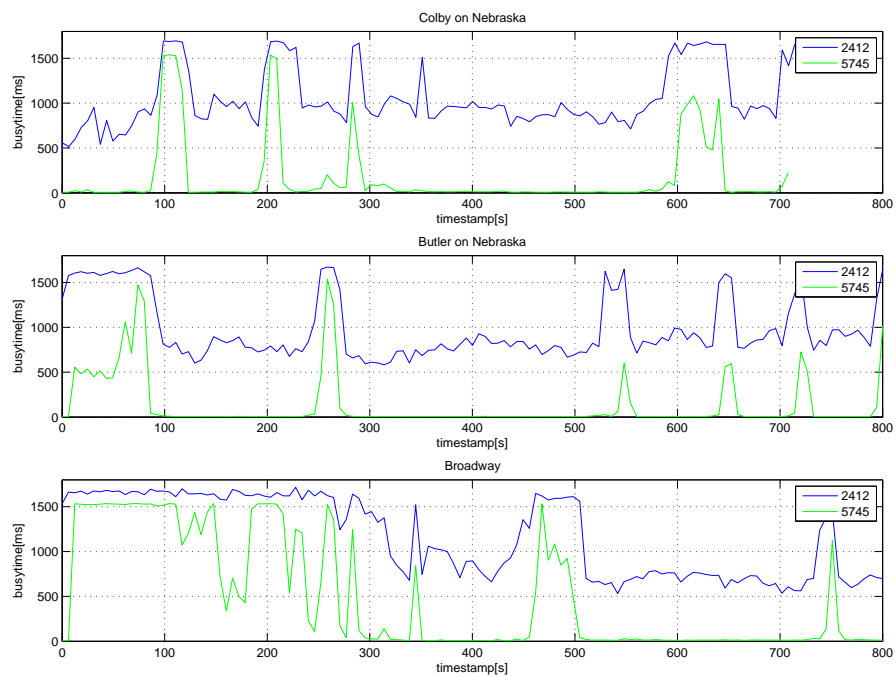


(b) Residential

FIGURE 8.4: Coverage Percent - 6Mbps - 10MHz



(a) Downtown



(b) Residential

FIGURE 8.5: Busy Channel Monitoring

Chapter 9

Conclusion and future work

This thesis proposes a control architecture to handle MAClets over programmable wireless nodes in order to implement dynamic adaptation policies in WLANs. Our framework extends the Wireless MAC Processor abstractions with frame classifiers, meta state machines and control messages, for effectively specifying and disseminating high-level hardware-independent policies. According to our vision, the proposed solution is an example of wireless software-defined network, where end users, network administrators and service providers can manage traffic flows over virtual and physical interfaces with customized access rules for mitigating the interference, reducing channel wastes and improving the overall network performance. Differently from the solutions currently explored in the wired domain, the introduction of state-dependent programs allow to natively implement reactive mechanisms, able to reprogram the node behaviors at the occurrence of pre-defined critical events. Besides the specific technical contributions, we believe that a further significance of our proposed approach is that protocol reconfiguration is accomplished via application programming interfaces, rather than via binary images or access to open source devices, thus perhaps permitting its possible future endorsement also in the real commercial world.

Appendix A

MAClet Manager

MAClet Manager is the software tool that define the Node interface between WMP and MAClet Controller. MAClet Manager is able to receive, send, dissect a MAClet Message, execute local WMP activation and injection bytecode commands. MAClet Manager conserve Bytecode Manager features and introduce MAClet format interpretation in client/server mode. MAClet Manager is written in C for GNU/Linux Systems and use `wmplib.h` library functions for broadcom B43 chipset interaction. The only working issue is to check that driver module has enabled debugfs, because `wmplib` currently works only with debugfs.

In this chapter a description of main commands and the features provided by MAClet Manager.

A.1 maclet-manager options list

```
root@alix3:~# ./maclet-manager
-----
MAClet Manager V 0.1 - 2012
-----
WMP maclet-manager byte-code injection
Usage: maclet-manager [OPTIONS]
```

```

-h          Print this help text
-l #        LOAD Bytecode in # memory area (1 or 2)
-m name-file LOAD Bytecode state-machine bytecode file
-n name-file LOAD Bytecode Only Parameter bytecode file
-a #        Activate specified bytecode (1 or 2)
-t time-value Timed Bytecode Activation [value in sec]
-d          Delayed Bytecode Activation in microsecond
-f time-value Return the absolute time for precise equal activation
            [value in sec]
-r          reset activation and deactivation condition Bytecode
-v          view dump information of bytecode variables
-c ip-addr  IP address to server station Start in client mode
-g name-file bytecode to send
-s          SERVER MODE
-p          In server mode or client mode select specific port,
            default 9898
-w          Template Ram Frame forging; frame can be 'date' or
            'ack' with different rate to the transmission, and
            string contained in the frame

```

A.2 Stand-alone Operation Mode

This section describes the following stand-alone functions:

- Bytecode injection and activation
- timed activation, delayed activation

A.2.1 Bytecode injection and activation

The main feature of the maclet-manager is Bytecode injection. A Bytecode can be injected into one of two different Bytecode areas: though area 1 is filled at startup with a default Bytecode, it can be replaced with a new one.

This command injects the Bytecode stored in file `mycode.txt` into area 2:

```
root@sta01# maclet-manager -l 2 -m mycode.txt
```

and this one activates it:

```
root@sta01# maclet-manager -a 2
```

To activate back the Bytecode in area 1:

```
root@sta01# maclet-manager -a 1
```

This command change the parameter Bytecode stored in file `myparameter.txt` into Bytecode located at the area 2:

```
root@sta01# maclet-manager -l 2 -n myparameter.txt
```

the option `-u` if used with the previous option force the load and activation in any case of lock.

A.2.2 Delayed Bytecode switching

Bytecode switching can be scheduled at a given time in the future, by either defining a delay or an absolute time: in both cases the event is handled by the WMP by periodically checking the internal clock. Since all stations in a given BSS synchronize their internal clock with that of the Access Point, the second mechanism allows to switch the Bytecode on several station at the same time.

This command schedules a Bytecode switch after twenty seconds:

```
root@sta01# maclet-manager -t 20
```

This command schedules a Bytecode switch at a given time:

```
root@sta01# maclet-manager -d <value-time-us>
```

where `value-time-us` is an accurate clock reference expressed in microsecond. When the internal clock reaches `value-time-us`, the WMP deactivates the current active Bytecode and activate the other one.

Again `maclet-manager` can be used to get the `value-time-us` corresponding to a given delay:

```
root@sta01# maclet-manager -f <delay-in-second>
```

The output value is expressed in microseconds and is computed by summing the input `delay-in-second` to the internal clock. For example, if we want to switch the Bytecode on all stations in 12 seconds we should first get the reference time on one station, i.e.,

```
root@sta01# maclet-manager -f 12
```

```
-----
MAClet Manager V 0.1 - 2012
-----
```

```
Selected find absolute time
Current work mode : "local"
-----
```

```
Calculation value of activation delay
time stamp : 3076057456
-----
```

Then we must run option `-d` on all stations using the `time stamp` value that was returned (3076057456).

To cancel timers, run:

```
root@sta01# maclet-manager -r
```

To display information about timers run, bytecode activated and register value:

```
root@alix2:~# ./maclet-manager -v
-----
MAClet Manager V 0.1 - 2012
-----
Current work mode : "local"
Selected view

-----

WMP INFORMATION

CURRENT BYTECODE                = 1
Control Value                    = 0x4000
Timer Not Active
Delay Not Active
-----

-----

REGISTER AND MEMORY INFORMATION

Current contention windows       = 0x001F
Max contention windows           = 0x01FF
Min contention windows           = 0x001F
Register 1                      = 0x0000
Register 2                      = 0x0000
Memory 1                        = 0x0000
Memory 2                        = 0x0000
Memory 3                        = 0x0000
-----
```

A.3 Client-Server Operation Mode

This section describes how to setup a Client-Server WMP configuration service using the MAClet Manager tool. First, the server should be started on a WMP station (e.g., `sta01`):

```
root@sta01# maclet-manager -s
-----
```

```

MAClet Manager V 0.1 - 2012
-----
Current work mode : "server"
Starting MAClet Manager in SERVER mode, listen on port 9898

```

By default TCP port 9898 is used but can be optionally changed with option `-p`. Once the server is running on `sta01`, all commands described in section [A.2](#) can be run from another machine (e.g. `sta02`):

```

root@sta02# maclet-manager -c sta01 -v

-----
MAClet Manager V 0.1 - 2012
-----
Current work mode : "client"
recvBuffer = |v=OK
-----
CURRENT BYTECODE = 1
Control Value    = 0x0000
Timer Not Active
Delay Not Active
-----

```

Also Bytecode images can be transferred using the client-server model: to send a Bytecode file table to a remote machine use this command:

```

root@sta02# maclet-manager -c sta01 -g /path/to/mybytecode.txt

-----
MAClet Manager V 0.1 - 2012
-----
file /path/to/mybytecode.txt found
Current work mode : "client"
filename sent=dcf_fix_bk.txt
recvBuffer = OK-FIN

```

Once the Bytecode is transferred it can be remotely loaded and activated: e.g.,

```

root@sta02# maclet-manager -c sta01 -l 2 -m mybytecode.txt

-----
MAClet Manager V 0.1 - 2012
-----
Current work mode : "client"
recvBuffer = |l=OK|m=OK

```

The MAC-Engine implements blocking techniques during Bytecode loading but, in some case, is necessary to force Bytecode injection without any safety procedure. MAClet Manager provides an option that allows hard injection of the Bytecode, useful when a Bytecode doesn't work correctly and, for example, loops in dead events. The option is `-u`

```
root@sta02# maclet-manager -c sta01 -l 2 -m mybytecode.txt -u
```

Finally, there are a few options to debug the WMP, by getting a dump of the MAC-Engine registers:

```
root@sta02# maclet-manager -x 1
-----
MAClet Manager V 0.1 - 2012
-----
Current work mode : "local"
Link registers:
lr0: 0AD1 lr1: 0B11 lr2: 0149 lr3: 0261
Offset registers:
off0: 0180 off1: 0204 off2: 039F off3: 039F
off4: 0130 off5: 045F off6: 3010
General purpose registers:
r00: 0001 r01: 0000 r02: 0001 r03: 000F
r04: 0000 r05: 0020 r06: 0007 r07: 0004
r08: 001F r09: 0000 r10: 0001 r11: 053E
[...cut...]
r52: 0001 r53: 000F r54: 01FF r55: 0000
r56: 0441 r57: 0418 r58: 0000 r59: 0708
r60: 0000 r61: 0000 r62: 0000 r63: 0000
```

and of the MAC-Engine memory:

```
root@sta02# maclet-manager -x 2
maclet-manager -x 2
-----
MAClet Manager V 0.1 - 2012
-----
Current work mode : "local"
SHM dump 2

Shared memory:

0x0000: 9A01 7008 FFFF 0A7C 0000 0000 C000 0A00
0x0010: 1400 0000 8000 0900 4700 4700 8301 6400
0x0020: 3009 C0FC 0000 0000 0000 0000 0000 0000
[...cut...]
```

To get a snapshot of the evolution of the state trace, first activate it:

```
root@sta02# maclet-manager -e on
-----
MAClet Manager V 0.1 - 2012
-----
Current work mode : "local"
Selected state-debug
```

Then dump the MAC-Engine memory and check memory in range [0x0E00 -0x0F60].

```
0x0E00: 0000 0000 0000 0000 0000 0000 0000 0000
0x0E10: 0000 FF00 0D00 FF00 FF08 0500 FF09 0900
0x0E20: FF00 0D00 FF00 0100 FF08 0500 FF09 0900
[...cut...]
0x0F30: 0100 FF08 0500 FF09 0900 FF00 0D00 FF00
0x0F40: 0100 FF08 0500 FF09 0900 FF00 0D00 FF00
0x0F50: 0100 FFFF 0000 0000 0000 0000 0000 0000
```

A.3.1 Forge template frame operation

This section describes the option used for forge a template frame that is possible send with specific action from WMP.

```
root@alix2:~# ./maclet-manager -w
-----
MAClet Manager V 0.1 - 2012
-----
Current work mode : "local"
Write frame into template ram

insert length frame : 200
[1] - 1Mbps
[6] - 6Mbps
[12] - 12Mbps
[18] - 18Mbps
[24] - 24Mbps
[36] - 36Mbps
[48] - 48Mbps
[54] - 54Mbps
insert rate : 24
insert destination address[12:34:56:78:9a:bc] = 12:34:56:78:9a:bd
insert frame string text : do
Write template frame success
```

There are specifics actions and events in to api WMP that are used for send a frame that is stored in the template ram of the device, is possible change many field of this template frame with the option -w, the field that is possible modify are:

- length frame
- rate
- destination address
- frame string text

A.3.2 Collect data

This section describes the option used to set and reset a specific number of registers and locations of memory used from WMP for collecting data.

There are specific actions and events in the API WMP that are used for set and reset 2 registers and 3 memory locations, it is possible to change the value of these registers also by the *maclet-manager* with the usage of option **-o**, the option follows the user step by step such as shown below.

```
root@alix2:~# ./maclet-manager -o
-----
MAClet Manager V 0.1 - 2012
-----
Current work mode : "local"
Managed other option

SET or RESET value for register and memory
[1] - REGISTER_1
[2] - REGISTER_2
[3] - MEMORY_1
[4] - MEMORY_2
[5] - MEMORY_3
insert option for set or reset : 1
[1] - RESET
[2] - SET
insert operation : 2
insert value to set : 43
shmWrite16 --- value = 2B
Write register success
```

Appendix B

Hardware Equipment

B.1 Introduction

In this chapter a description of hardware equipment adopted for development, testing and experiment setup. All Experiment have been conducted using ISM 802.11 Channel spectrum adopting a commodity broadcom b43xx WLAN card. Node devices equipped with b43xx are able to work with WMP and exploit MAC program loading/execution, MAClet communication and MAClet Controller interactions.

B.2 WLAN 802.11 NIC - Broadcom BCM4311 - BCM4318

AirForce™ 54g® 802.11b/g PCI Express® Transceiver

Designed for client devices based on the PCI Express architecture, the Broadcom BCM4311 is a 802.11b/g baseband processor with an integrated media access controller (MAC).¹ In figure B.2 a block diagram description. The Broadcom Processor for BCM4311 or BCM4318 works with the same firmware version at system sight we have no differences. In most usecase has been used BCM4318 because it has a miniPCI port interface compatible with Alix 2D2.

This Chip is equipped with a 4K SPROM memory that can be used for customer designed parameters. WMP prototype use a part of this memory for MAC program and WMP parameter injecting.

¹<http://www.broadcom.com/products/Wireless-LAN/802.11-Wireless-LAN-Solutions/BCM4311>

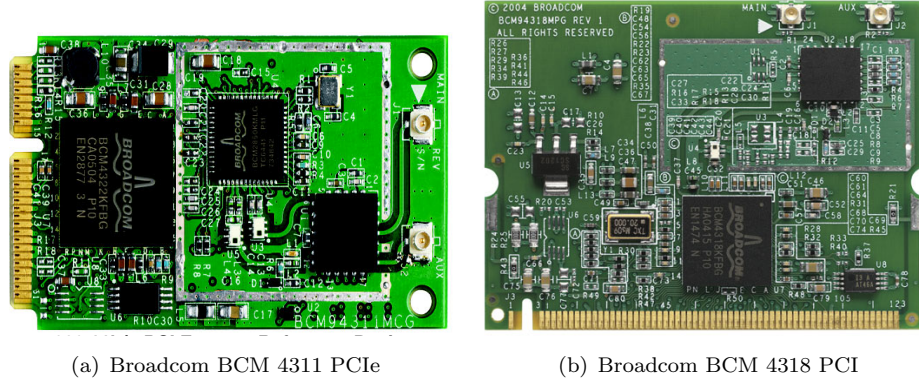
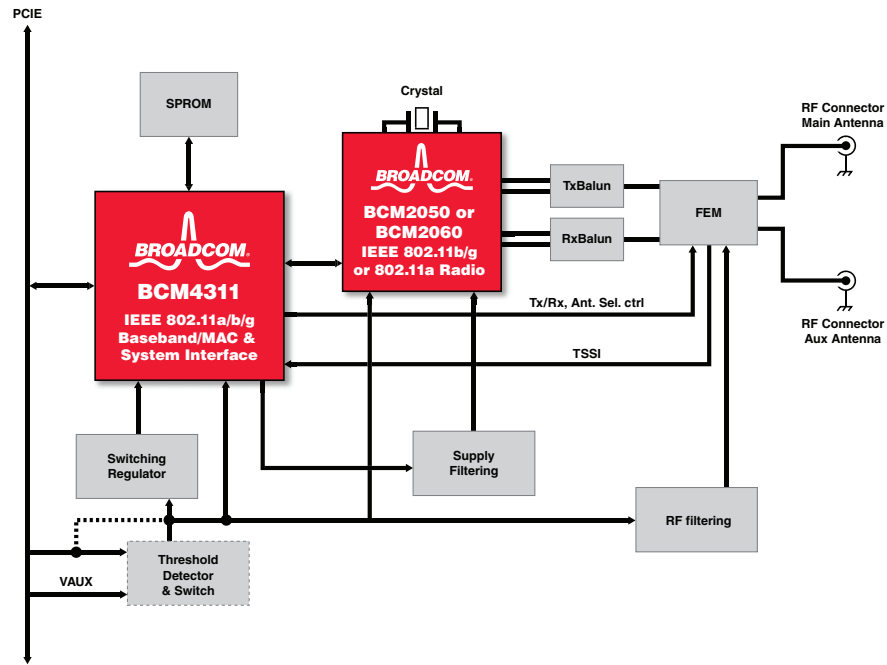


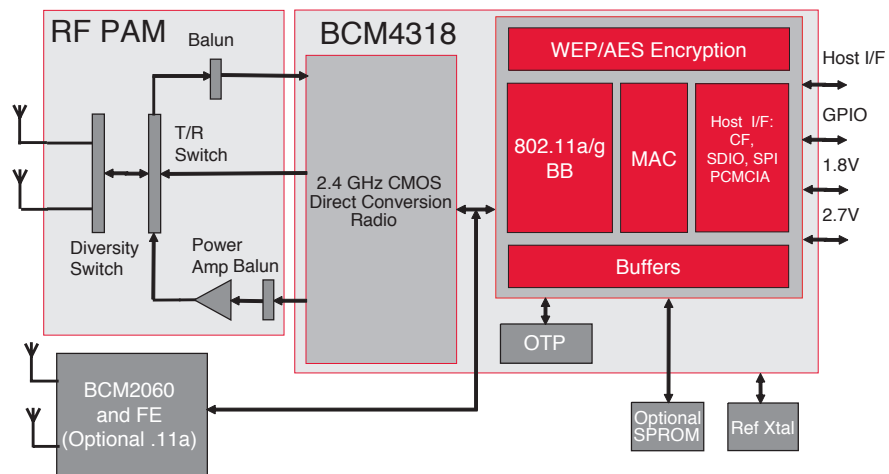
FIGURE B.1: BCM43xx NICs

Interface	USB 1.1 Host interface, PCIe™, GPIO, UART, JTAG
System bus support Standard	PCIe IEEE 802.11g with external radio BCM2050, 802.11a with external radio BCM2060
Data rate	802.11g: 54, 48, 36, 24, 18, 12, 9, 6 Mbps; 802.11b, 11, 5.5, 2, 1 Mbps
Modulation	OFDM, CCK, DQPSK, DBPSK
Network architectures	Infrastructure and ad hoc
Operating frequencies	2.4 GHz—2.497 GHz
Operating channels	11 for North America; 13 for Europe, and 14 for Japan
RF output power	20 dBm maximum
Antenna connectors	Hardware diversity support—Transmit and Receive
Power requirements	3.3V for reference designs
Security	802.1x; WEP; WEP2; WPA, WPA2; TKIP; WEP128; Weak-key avoidance; CCX, CCX 2.0; 128-bit OCB mode AES, 802.11i
Certifications	IEEE 802.11 compliant; WI-FI CERTIFIED™; ACPI power management

TABLE B.1: Broadcom BCM4311 Specification



(a) Broadcom BCM 4311 Block Diagram



(b) Broadcom BCM 4318 Block Diagram

FIGURE B.2: Broadcom BCM 43xx Block Diagram

B.3 Wireless Nodes, PC Engines Alix Board

Alix Board Specifications	
alix2d2	System board
Part numbers	2 LAN 2 miniPCI LX800 256 MB USB
Spec	<ul style="list-style-type: none"> • CPU: 500 MHz AMD Geode LX800 • DRAM: 256 MB DDR DRAM • Storage: CompactFlash socket, 44 pin IDE header • Power: DC jack or passive POE, min. 7V to max. 20V • Three front panel LEDs, pushbutton • Expansion: 2 miniPCI slots, LPC bus • Connectivity: 2 Ethernet channels (Via VT6105M 10/100) • IO: DB9 serial port, dual USB port • Board size: 6 x 6" (152.4 x 152.4 mm) - same as WRAP.1E • Firmware: tinyBIOS
Manufacturer	PC Engines

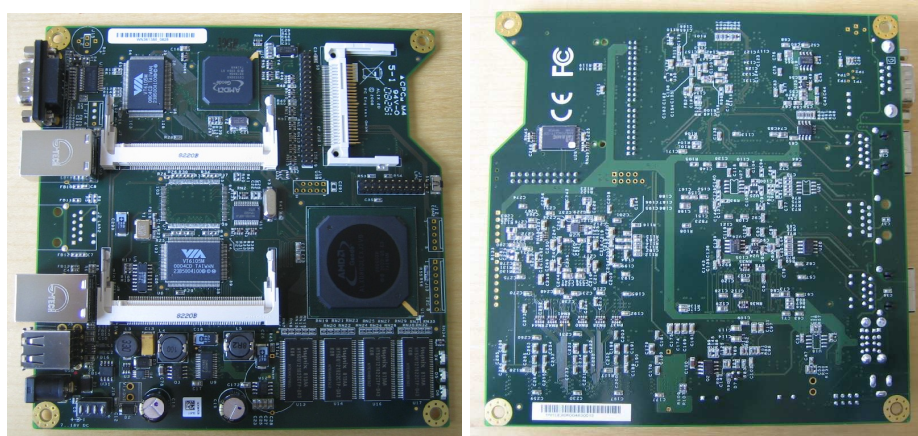


FIGURE B.3: Alix 2D2

System Settings	
OS	GNU/Linux Ubuntu 11.10, kernel 3.0.17
WMP	ucode5.fw installed in /lib/firmware/b43/ directory
MAClet Manager	Installed

B.4 USRP

for a physical channel occupancy evaluation have been used Ettus USRP and USRP2 ²

The primary driver for all Ettus Research products - including the USRP2 - is the UHD (USRP Hardware Driver) software. UHD software is considered stable and is actively maintained by Ettus Research. The UHD driver is recommended for all users.

²<http://gnuradio.org/redmine/projects/gnuradio/wiki/USRP2>

The original driver exists for the USRP2 that uses a raw Ethernet implementation is still available within GNU Radio. The older driver is no longer maintained and not recommended for use. Legacy driver users should upgrade firmware and drivers to UHD software.

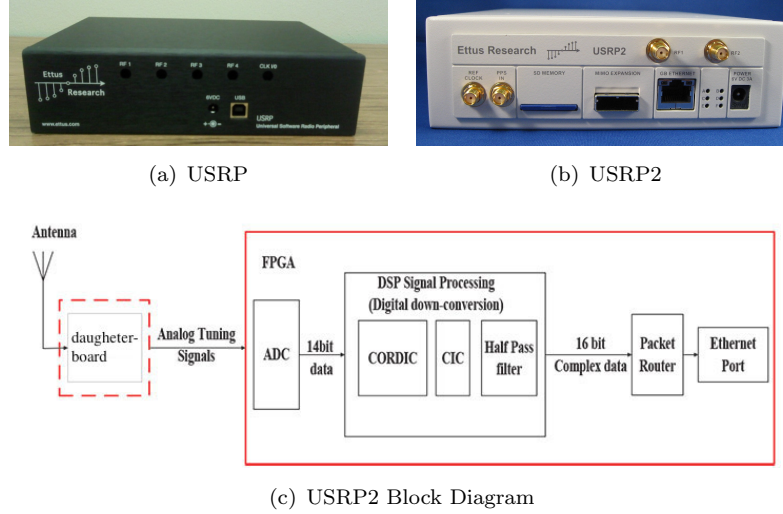


FIGURE B.4: USRP1 USRP2 frontend and USRP2 block diagram

The USRP2 builds on the success of the original USRP device, and adds these new features:

- Gigabit Ethernet interface
- 25 MHz of instantaneous RF bandwidth
- Xilinx Spartan 3-2000 FPGA
- Dual 100 MHz 14-bit ADCs
- Dual 400 MHz 16-bit DACs
- 1 MByte of high-speed SRAM
- Locking to an external 10 MHz reference
- 1 PPS (pulse per second) input
- Configuration stored on standard SD cards
- Standalone operation
- The ability to lock multiple systems together for MIMO
- Compatibility with all the same daughterboards as the original USRP

B.4.1 ADC/DAC section

The received signal is sampled by the ADC and converted to digital values depending on the ADCs dynamic range.

In the USRP1, there are 4 high-speed 12-bit AD converters. The sampling rate is 64M samples per second. In principle, it could digitize a band as wide as 32MHz. The AD converters can bandpass-sample signals of up to about 200MHz. If several decibels of loss is tolerable, then, IF frequency as high as 500 MHz can be digitized. However,

if we sample a signal with the IF larger than 32MHz, we will introduce aliasing and actually the band of the signal of interest is mapped to some places between -32MHz and 32MHz. Sometimes this can be useful, for example, we could listen to the FM stations without any RF front end. The higher the frequency of the sampled signal, the more the SNR will be degraded by jitter. 100MHz is the recommended upper limit. The USB connection sustains 32 MB/s in half duplex, so transmission and reception of samples isn't possible synchronously. The full range of the ADCs is 2V peak to peak, and the input is 50 ohms differential. This is 40mW, or 16dBm. There is a programmable gain amplifier (PGA) before the ADCs to amplify the input signal to utilize the entire input range of the ADCs, in case the signal is weak. The PGA is up to 20dB. With gain set to zero, full scale inputs are 2 Volts peak-to-peak differential. When set to 20 dB, only .2 V p-p differential input signal is needed to reach full scale. This PGA is software programmable. If signals are AC-coupled, there is no need to provide DC bias as long as the internal buffer is turned on. It will provide an approximately 2V bias. If signals are DC-coupled, a DC bias of $V_{CC}/2$ (1.65V) should be provided to both the positive and negative inputs, and the internal buffer should be turned off. The ADC VREF provides a clean 1 V reference.

At the transmitting path, there are also 4 high-speed 14-bit DA converters. The DAC clock frequency is 128 MS/s, so Nyquist frequency is 64MHz. However, we will probably want to stay below it to make filtering easier. A useful output frequency range is from DC to about 44MHz. The DACs can supply 1V peak to a 50 ohm differential load, or 10mW (10dBm). There is also PGA used after the DAC, providing up to 20dB gain. This PGA is software programmable. The DAC signals (IOUTP_A/IOUTN_A and IOUTP_B/IOUTN_B) are current-output, each varying between 0 and 20 mA. They can be converted into differential voltages with a resistor.

The USRP2 use a Dual 14-bit LTC2284 at 100MS/s as its main ADC. There is an auxiliary 2 channels, 12-bit ADC (the AD7922) for each daughterboard connector. The main DAC is the Dual 16-bit AD9777 fed with 100 Ms/s and produces 400 Ms/s based analog output. The auxiliary DACs are the dual 12-bit AD5623. As we can see, the Gb-Ethernet allow a significant higher throughput for the USRP2 compared to USRP1 with USB 2.0 and the theoretically data rate of 125 MB/s allows for a theoretical (complex) RF bandwidth of about 31,25 MHz.

B.4.2 The FPGA

FPGA “plays a key role in the GNU Radio system” (according to GnuRadio documentation). The FPGA is like a small, massively parallel computer that performs high bandwidth math to reduce the data rates to something we can manageably transfer over the USB2.0 link (USRP1) or Gb-Ethernet (USRP2). With the USRP1, it can only be reprogrammed over the USB2 bus, and with the USRP2 the FPGA has to be loaded as well as the firmware on the SD Card.

The FPGA includes a Digital Down-Converter (DDC) implemented with Cascaded Integrator-Comb (CIC) filters, which are very high-performance filters that use only adds and delays. The purpose of DDCs is to mix the signal to a lower frequency and reduce the sample rate while retaining all the information. ³

³http://inviertez.handgrip.se/doc/Introduction_to_USRP

B.4.3 Usage Example

Transmission Traces are acquired using `uhd_rx_cfile` command. The follow script snippet provide an example of acquisition USRP command parameters.

```
CHANNEL=6
GAIN=28
MILLISECONDS=30000
DECIM=8
FREQ=$((2412+5*({CHANNEL}-1)))
RATE=25

uhd_rx_cfile --samp-rate ${RATE}M -f ${FREQ}M -g $GAIN -s \
-N $((($RATE*1000*${MILLISECONDS})) output_file.raw
```

USRP2 has been used for evaluation experiments as a physical traffic analyzer able to measure the interarrival time between frames and Time Division Access Scheme debugging. In figure B.5 an output example of a multiple node transmissions in IEEE 802.11 DCF.

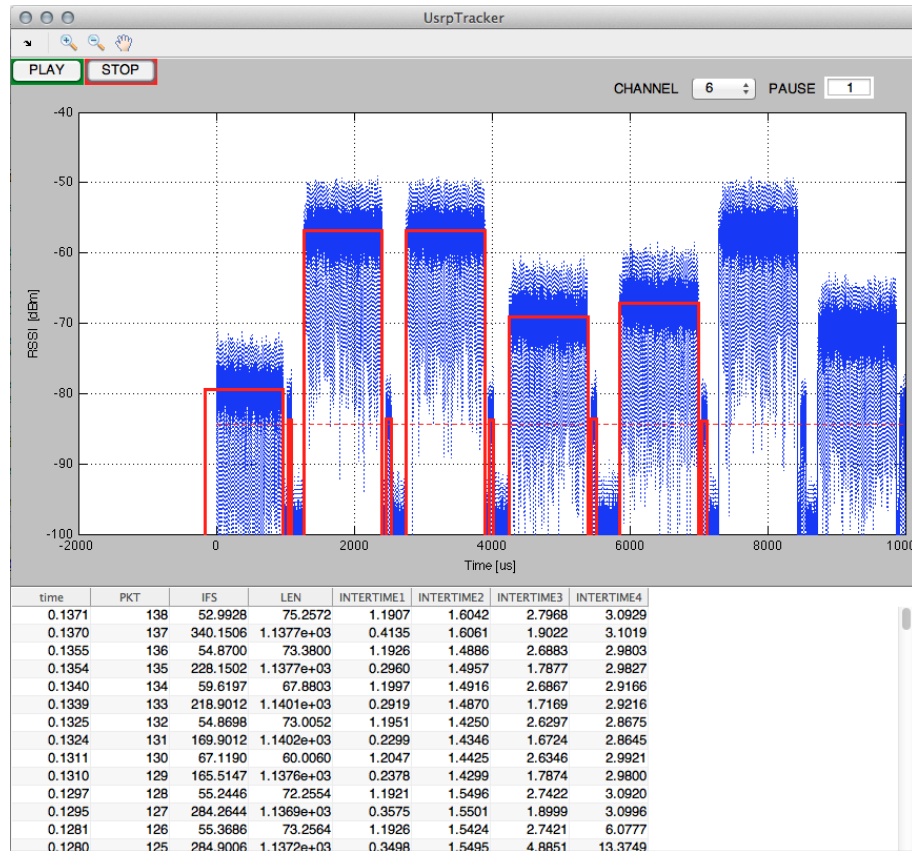


FIGURE B.5: USRP Tracker interactive tool

Bibliography

- [1] CalRadio - <http://calradio.calit2.net/calradio1.htm>.
- [2] ETHANE - <http://yuba.stanford.edu/ethane/>.
- [3] Linux Wireless Kernel Project - <http://wireless.kernel.org/>.
- [4] Linux Wireless Documentation - <http://wireless.kernel.org/en/developers/Documentation/mac80211>.
- [5] NOX Project - <http://www.noxrepo.org/>.
- [6] OMF - <http://oml.mytestbed.net/projects/oml/wiki>.
- [7] OML - <http://www.cs.berkeley.edu/~ananthar/oml.html>.
- [8] ONF - <https://www.opennetworking.org/>.
- [9] ONF white paper. Software-Defined Networking: The New Norm for Networks.
- [10] Open firmware for WiFi networks.
- [11] OpenFlow Specification v1.0.0.
- [12] Pantou : OpenFlow 1.0 for OpenWRT.
- [13] POX - <http://www.noxrepo.org/pox/about-pox/>.
- [14] Pox wiki webpage - <https://openflow.stanford.edu/display/onf/pox+wiki>.
- [15] Project 4D - <http://www.cs.cmu.edu/~4D/>.
- [16] RFC3292 - <http://www.ietf.org/rfc/rfc3292.txt>.
- [17] The GNURadio Software Radio - <http://gnuradio.org/trac>.
- [18] USRP. The universal software radio peripheral.
- [19] Wireless Open Access Research Platform.
- [20] M. NEUFELD, J. FIFIELD, C. DOERR, A. SHETH, D. GRUNWALD. SoftMAC - Flexible Wireless Research Platform. *HotNets* (Nov. 2005).
- [21] A. DI STEFANO G. TERRAZZINO C. GIACONIA. FPGA Implementation of a Re-configurable 802.11 Medium Access Control, April 2006.

- [22] AMEIXEIRA, C., MATOS, J., MOREIRA, R., CARDOTE, A., OLIVEIRA, A., AND SARGENTO, S. An IEEE 802.11p/WAVE implementation with synchronous channel switching for seamless dual-channel access (poster). In *VNC (2011)*, O. Altintas, W. Chen, and G. J. Heijenk, Eds., IEEE, pp. 214–221.
- [23] ANCILLOTTI, E., BRUNO, R., CONTI, M., AND PINIZZOTTO, A. Dynamic address autoconfiguration in hybrid ad hoc networks. *Pervasive and Mobile Computing* 5, 4 (2009), 300–317.
- [24] ANDREW T. CAMPBELL AND IRENE KATZELA. *Open Signaling For Atm, Internet And Mobile Networks (Opensig'98)*. 1999.
- [25] ARSLAN, M. Y., PELECHRINIS, K., BROUSTIS, I., KRISHNAMURTHY, S. V., AD-DEPALLI, S., AND PAPAGIANNAKI, K. Auto-configuration of 802.11n WLANs. In *CoNEXT (2010)*, J. C. de Oliveira, M. Ott, T. G. Griffin, and M. Médard, Eds., ACM, p. 27.
- [26] BERNASCHI, M., CACACE, F., IANNELLO, G., VELLUCCI, M., AND VOLLERO, L. OpenCAPWAP: an open-source CAPWAP implementation for management and QoS support. *IT-NEWS* (2008).
- [27] BIANCHI, G., GALLO, P., GARLISI, D., GIULIANO, F., GRINGOLI, F., AND TINNIRELLO, I. MAClets: active MAC protocols over hard-coded devices. In *CoNEXT (2012)*, C. Barakat, R. Teixeira, K. K. Ramakrishnan, and P. Thiran, Eds., ACM, pp. 229–240.
- [28] BIANCHI, G., GALLO, P., GARLISI, D., GIULIANO, F., GRINGOLI, F., AND TINNIRELLO, I. A Control Architecture for Wireless MAC Processor Networking.
- [29] BOHGE, M., GROSS, J., AND WOLISZ, A. Optimal soft frequency reuse and dynamic sub-carrier assignments in cellular OFDMA networks. *European Transactions on Telecommunications* 21, 8 (2010), 704–713.
- [30] BOSE, V. G., WETHERALL, D., AND GUTTAG, J. V. Next Century Challenges: RadioActive Networks. In *MOBICOM (1999)*, H. Kodesh, V. Bahl, T. Imielinski, and M. Steenstrup, Eds., ACM, pp. 242–248.
- [31] C. DOERR, M. NEUFELD, J. FIFIELD, T. WEINGART, D.C. SICKER, D. GRUNWALD. MultiMAC - An adaptive MAC Framework for Dynamic Radio Networking. *IEEE DySPAN 2005* (Nov. 2005).
- [32] CAMPBELL, A. T., DE MEER, H. G., KOUNAVIS, M. E., MIKI, K., VICENTE, J. B., AND VILLELA, D. A Survey of Programmable Networks. *SIGCOMM Comput. Commun. Rev.* 29, 2 (Apr. 1999), 7–23.
- [33] COSTANZO, S., GALLUCCIO, L., MORABITO, G., AND PALAZZO, S. Software Defined Wireless Networks: Unbridling SDNs. *2013 Second European Workshop on Software Defined Networks 0* (2012), 1–6.
- [34] GARLISI, D., GIULIANO, F., TINNIRELLO, I., GALLO, P., GRINGOLI, F., AND G., B. Deploying Virtual MAC Protocols Over a Shared Access Infrastructure Using MACLETS.
- [35] GRAYSON, M., AND TAVARES, J. System and method for providing resource management in a network environment, May 21 2013. US Patent 8,447,314.

- [36] GRINGOLI, F., FACCHI, N., GALLO, P., GARLISI, D., GIULIANO, F., AND BIANCHI, G. Enabling Cognitive-radio Paradigm on Commercial Off-the-shelf 802.11 Hardware. In *Proceedings of the 8th ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization* (2013), WiNTECH '13, ACM, pp. 95–96.
- [37] HU, H., ZHANG, J., ZHENG, X., YANG, Y., AND WU, P. Self-configuration and Self-optimization for LTE Networks. *Comm. Mag.* 48, 2 (Feb. 2010), 94–100.
- [38] I. TINNIRELLO G. BIANCHI P. GALLO D. GARLISI F. GIULIANO F. GRINGOLI. Wireless MAC Processors: Programming MAC Protocols on Commodity Hardware. 1–9.
- [39] INSTITUTE, C. M. R. C-RAN — The Road Towards Green RAN. *White Paper* (Apr 2010).
- [40] KAUFFMANN, B., BACCELLI, F., CHAINTREAU, A., MHATRE, V., PAPAGIANNAKI, K., AND DIOT, C. Measurement-Based Self Organization of Interfering 802.11 Wireless Access Networks. In *INFOCOM* (2007), IEEE, pp. 1451–1459.
- [41] KIM, H., AND FEAMSTER, N. Improving network management with software defined networking. *IEEE Communications Magazine* 51, 2 (2013), 114–119.
- [42] KRASNIEWSKI, M. D., PANTA, R. K., BAGCHI, S., YANG, C.-L., AND CHAPPELL, W. J. Energy-efficient on-demand reprogramming of large-scale sensor networks. *TOSN* 4, 1 (2008).
- [43] LEVANTI, A., GIORDANO, F., AND TINNIRELLO, I. A CAPWAP-Compliant Solution for Radio Resource Management in Large-Scale 802.11 WLAN. In *GLOBECOM* (February 2009), IEEE, pp. 3645–3650.
- [44] LEVIS, P., AND CULLER, D. Mate: A Tiny Virtual Machine for Sensor Networks, 2002.
- [45] LEVIS, P., PATEL, N., CULLER, D., AND SHENKER, S. Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. In *In Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)* (2004), pp. 15–28.
- [46] LI, L. E., MAO, Z. M., AND REXFORD, J. Toward Software-Defined Cellular Networks. *2013 Second European Workshop on Software Defined Networks 0* (2012), 7–12.
- [47] LOBINGER, A., STEFANSKI, S., JANSEN, T., AND BALAN, I. Coordinating Handover Parameter Optimization and Load Balancing in LTE Self-Optimizing Networks. In *VTC Spring* (2011), IEEE, pp. 1–5.
- [48] LU, M.-H., STEENKISTE, P., AND CHEN, T. Using commodity hardware platform to develop and evaluate CSMA protocols. In *WINTECH* (2008), A. Sabharwal, R. Karrer, and L. Zhong, Eds., ACM, pp. 73–80.
- [49] LUCENT, A. Lightradio portfolio - technical overview. *White Paper*.
- [50] MADDEN, S. R., FRANKLIN, M. J., HELLERSTEIN, J. M., AND HONG, W. TinyDB: An Acquisitional Query Processing System for Sensor Networks. *ACM Trans. Database Syst.* 30, 1 (Mar. 2005), 122–173.

- [51] MAYER, C. P., AND WALDHORST, O. P. Offloading infrastructure using Delay Tolerant Networks and assurance of delivery. In *Wireless Days* (2011), IEEE, pp. 1–7.
- [52] McKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., PARULKAR, G., PETERSON, L., REXFORD, J., SHENKER, S., AND TURNER, J. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.* 38, 2 (2008), 69–74.
- [53] MITOLA, J., AND MAGUIRE, G.Q., J. Cognitive radio: making software radios more personal. *IEEE Personal Communications* 6, 4 (1999), 13–18.
- [54] MOGUL, J. C. AND YALAGANDULA, P., TOURRILHES, J., McGEER, R., BANERJEE, S., CONNORS, T., AND SHARMA, P. Orphal: Api design challenges for open router platforms on proprietary hardware.
- [55] MURTY, R., PADHYE, J., CHANDRA, R., WOLMAN, A., AND ZILL, B. Designing high performance enterprise wi-fi networks. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2008), NSDI'08, USENIX Association, pp. 73–88.
- [56] MURTY, R., PADHYE, J., WOLMAN, A., AND WELSH, M. An Architecture for Extensible Wireless LANs. In *Proc. Seventh ACM Workshop on Hot Topics in Networks (HotNets-VII)* (October 2008).
- [57] NG, M. C. A., FLEMING, K. E., VUTUKURU, M., GROSS, S., ARVIND, AND BALAKRISHNAN, H. Airblue: A System for Cross-Layer Wireless Protocol Development. In *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)* (La Jolla, CA, October 2010).
- [58] P. DJUKIC AND P. MOHAPATRA. Soft-TDMAC: A Software-Based 802.11 Overlay TDMA MAC with Microsecond Synchronization. *IEEE Transactions on Mobile Computing* 11, 3 (2012), 478–491.
- [59] PANTA, R. K., KHALIL, I. M., AND BAGCHI, S. Stream: Low Overhead Wireless Reprogramming for Sensor Networks. In *INFOCOM* (2007), IEEE, pp. 928–936.
- [60] PARK, S.-H., SIMEONE, O., SAHIN, O., AND SHAMAI, S. Robust and Efficient Distributed Compression for Cloud Radio Access Networks. *IEEE T. Vehicular Technology* 62, 2 (2013), 692–703.
- [61] RAHMAN, M., AND YANIKOMEROGLU, H. Enhancing cell-edge performance: a downlink dynamic interference avoidance scheme with inter-cell coordination. *IEEE Transactions on Wireless Communications* 9, 4 (2010), 1414–1425.
- [62] S. LIU, J. WU AND C.H. KOH AND V.K.N. LAU. A 25 Gb/s(/km²) urban wireless network beyond IMT-advanced. *IEEE Communications Magazine* 49, 2 (February 2011), 122–129.
- [63] SHRIVASTAVA, VIVEK AND AHMED, NABEEL AND RAYANCHU, SHRAVAN AND BANERJEE, SUMAN AND KESHAV, SRINIVASAN AND PAPAGIANNAKI, KONSTANTINA AND MISHRA, ARUNESH. CENTAUR: Realizing the Full Potential of Centralized Wlans Through a Hybrid Data Path. In *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking* (New York, NY, USA, 2009), MobiCom '09, ACM, pp. 297–308.

- [64] TAN, K., ZHANG, J., FANG, J., LIU, H., YE, Y., WANG, S., ZHANG, Y., WU, H., WANG, W., AND VOELKER, G. M. Sora: High Performance Software Radio Using General Purpose Multi-core Processors.
- [65] TEMPLIN, F. Virtual Enterprise Traversal (VET). RFC 5558.
- [66] TENNENHOUSE, D. L., SMITH, J. M., SINCOSKIE, W. D., WETHERALL, D. J., AND MINDEN, G. J. A Survey of Active Network Research. *IEEE Communications Magazine* 35 (1997), 80–86.
- [67] V. BOSE D. WETHERALL J. GUTTAG. Next century challenges: RadioActive networks. *ACM/IEEE MobiCom '99*, 242–248.
- [68] VENTURINO, L., PRASAD, N., AND WANG, X. Coordinated Scheduling and Power Allocation in Downlink Multicell OFDMA Networks. *IEEE T. Vehicular Technology* 58, 6 (2009), 2835–2848.
- [69] VOELLMY, A., AGARWAL, A., HUDAK, P., FEAMSTER, N., BURNETT, S., AND JULY, J. L. Don't configure the network, program it! Domain-specific programming languages for network systems. Tech. rep., 2010.
- [70] WETHERALL, D., GUTTAG, J., AND TENNENHOUSE, D. ANTS: A toolkit for building and dynamically deploying network protocols. In *Proceedings of IEEE Openarch'98* (Apr. 1998).
- [71] YAP, K.-K., KOBAYASHI, M., SHERWOOD, R., HUANG, T.-Y., CHAN, M., HANDIGOL, N., AND MCKEOWN, N. OpenRoads: empowering research in mobile networks. *Computer Communication Review* 40, 1 (2010), 125–126.

Publications

- P1. F. Gringoli, D. Garlisi, P. Gallo, F. Giuliano, S. Mangione, I. Tinnirello “MAC-Engine: a new architecture for executing MAC algorithms on commodity WiFi hardware” WINTech '11 Las Vegas, Nevada, USA Sep, 2011
- P2. I. Tinnirello, G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, F. Gringoli, “Wireless MAC Processors: Programming MAC Protocols on Commodity Hardware”, IEEE INFOCOM, March 2012.
- P3. P. Gallo, D. Garlisi, F. Giuliano, F. Gringoli, I. Tinnirello, “WMPS: A Positioning System for Localizing Legacy 802.11 Devices”, IEEE Transactions on Smart Processing and Computing, In Press.
- P4. D. Garlisi, F. Giuliano, I. Tinnirello, P. Gallo, F. Gringoli, G. Bianchi, “Deploying Virtual MAC Protocols Over a Shared Access Infrastructure Using MACLETS”, IEEE INFOCOM 2013 Workshop, Turin, Italy, April 14-19, 2013.
- P5. G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, F. Gringoli, I. Tinnirello, “MAClets: Active MAC Protocols over hard-coded devices”, ACM CoNEXT 2012, Nice, France, December 10-13, 2012.
- P6. G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, F. Gringoli, I. Tinnirello, “A Control Architecture for Wireless MAC Processor Networking”, Future Network and Mobile Summit 2013, Lisbon, Portugal, 03-05 July, 2013.
- P7. F. Gringoli, N. Facchi, P. Gallo, D. Garlisi, F. Giuliano, G. Bianchi “Enabling Cognitive-Radio Paradigm on Commercial Off-The-Shelf 802.11 Hardware”, ACM WINTech 2013, Miami, Florida USA, September 30th, 2013
- P8. Gallo, P.; Garlisi, D.; Giuliano, F.; Gringoli, F.; Tinnirello, I.; Bianchi, G. “Wireless MAC Processor Networking: A Control Architecture for Expressing and Implementing High-Level Adaptation Policies in WLANs”, IEEE Vehicular Technology Magazine, Dec 2013, volume: 8, issue:4, pages: 81 - 89