Future Network and MobileSummit 2013 Conference Proceedings
Paul Cunningham and Miriam Cunningham (Eds)
IIMC International Information Management Corporation, 2013
ISBN: 978-1-905824-36-6

A Control Architecture for Wireless MAC Processor Networking

Pierluigi GALLO[1], Domenico GARLISI[1], Fabrizio GIULIANO[1],
Francesco GRINGOLI[2], Ilenia TINNIRELLO[1] and Giuseppe BIANCHI[3],
[1]*CNIT / Università di Palermo, DEIM, Palermo, Italy*
Emails: {pierluigi.gallo, domenico.garlisi, fabrizio.giuliano, ilenia.tinnirello}@unipa.it
[2]*CNIT / Università di Brescia, DII, Brescia, Italy*
Email: francesco.gringoli@ing.unibs.it
[3]*CNIT / Università di Roma Tor Vergata, Italy*
Email: giuseppe.bianchi@uniroma2.it

Abstract: In these years, the proliferation of unplanned WLANs is creating the need of implementing different *adaptation* strategies for improving the network performance under mutating and evolving interference scenarios. Many vendors propose undisclosed MAC/PHY optimization solutions, such as ambient noise immunity schemes, dynamic tuning of operating channels and contention parameters, etc., relying on low-level implementations in the card hardware/firmware.
In this paper we envision a new solution for expressing and implementing high-level adaptation policies in WLANs, in contrast to the current approaches based on vendor-specific implementations. We exploit the hardware abstraction interface recently proposed by the Wireless MAC Processor (WMP) architecture, and some flow-control concepts similar to the Openflow model for defining MAC adaptation policies. A simple control architecture for disseminating and activating new policies among multiple nodes is validated in an experimental testbed.

Keywords: Control architecture, Wireless MAC Processor, MAClet, distribution, flow control

1. Introduction

In the last years it clearly emerged that, in order to support the increasing demand of mobile traffic, a greater level of adaptability has to be supported by wireless local networks for exerting a fine-grained and smart control of the radio resources. On one side, current technologies include several PHY layer enhancements, including multi antennas, channel bonding capabilities, adaptive modulations, etc., for optimizing the channel capacity according to the propagation and interference conditions. Dynamic spectrum access and cognitive technologies are also suggesting a new networking paradigm, with devices able to sense the environment and reprogram themselves in reaction to mutating topologies, spectrum conditions, or application requirements [1, 2]. On the other side, technical solutions for supporting device reconfigurations at layers higher than the PHY (and in particular on the MAC layer, as considered in this paper), are also emerging for supporting different MAC/PHY adaptation policies, such as ambient noise immunity solutions, dynamic tuning of contention parameters, multiple virtual interfaces on the same hardware [3], and so on. Most of these adaptation solutions are vendor-specific, focused on a particular mechanism, and with a pre-defined low-level implementation in the card hardware/firmware. An interesting problem is understanding if MAC adaptation can be *dynamic*, supported over *closed* devices with undisclosed hardware internal design, and defined on the basis of some *mechanism-independent* primitives. In fact,

these features could make networks easily configurable for users and operators. Assume for example that a network operator wants to switch from contention-based to time-division access in case of high load conditions. TDMA is obviously not supported by commercial WiFi cards, although overlay modules have been shown to emulate this access mechanism over open source modified drivers [4]. Apart from the technical feasibility of the new scheme, the operator should ask its associated stations to install the TDMA overlay module, and should implement a decision module on its Access Points and a signaling mechanism for asking its associated stations to run the overlay TDMA module. The operator could also decide to force all the stations to disable the rate adaptation scheme or to use the same ARF scheme, but it has no mean to inject on-the-fly this abstract policy into the associated stations (without entering into the card internals and modifying again the specific drivers). In this paper, starting from the recent advances in the definition of new card architectures able to support programmable MAC schemes, we consider the possibility to program and disseminate adaptation policies in a wireless local network. Our solution has some similarities with the Openflow model [5], which apply similar concepts to the wired domain, i.e. making the router forwarding policies programmable by means of open API, without requiring manufacturers to disclose the internals of the switch fabrics. In our wireless scenario, programming the forwarding policies is more complex because the unconfined nature of the wireless medium makes the selection of the best possible (real or virtual) network interface strictly related to the relevant medium access rules. Indeed, customized medium access rules can help in providing isolation or coordination among multiple links. We prove how the joint definition of control policies for *traffic flow classification* and *medium access* can be exploited by our Control Architecture in a real testbed referring to common home networking scenarios.

2. Related Work

Programmability and code mobility in the wireless communications has been explored along several directions [6], with particular attention to sensor networks [7, 8] as a solution for distributing upgrades. In this same field, [9] introduced a protocol for distributing the code, while [10] faced the problem of efficient distribution by replacing the concept of binary with interpreted code. These works limit programmability above the MAC layer and in any case differ from our vision, since the code is system-specific and no abstraction is used for simplifying its definition.

First attempts to demonstrate advanced MAC programming interfaces leveraged the openness of SoftMAC drivers on off-the-shelf 802.11 hardware [11, 12, 4]: though they enable the configuration of MAC registers including contention window(s), slot time, and transmit power, they do not allow precise time-constraint scheduling of customized frame, as such operations require changes in proprietary firmwares. The opensource firmware [13] released for Broadcom cards changed a bit the situation and led to challenging demonstrations [14] but requires deep skills in assembly coding and it is still far away from the flexibility requirements needed by active networking and code mobility. The research community tried circumventing these issues by resorting to fully programmable designs based either on Software Define Radios (SDRs) or FPGA circuits [15, 16]: however the former prohibit the implementation of time-constraint protocols because of high latencies, while the extreme versatility offered by the latter

clashes with the needs for offline reloading of Verilog and C compiled code.

A first step towards the definition of a simple Application Programming Interface for programming MAC schemes by re-using already developed functionalities is represented by [17, 18, 19]. Among these approaches we focused on the Wireless MAC Processor (WMP) architecture [19] that has been implemented over an ultra-cheap commercial card (by Broadcom), over which different access rules have been experimentally validated (including TDMA, CSMA and multi-channel access schemes).

2.1 The Wireless MAC Processor

For abstracting the hardware capabilities and defining hardware agnostic programs to be injected into the wireless cards, the WMP architecture implements a state machine execution engine, called MAC engine, and a set of elementary actions and signals directly on the card. The MAC Engine is an executor of ***MAC Programs***, specified in terms of an high-level state machine that can be ported on different card systems. The definition of the medium access control logic in terms of extended finite state machine (XFSM) permits to conveniently control the ***actions*** performed on the hardware, as a consequence of the MAC protocol logic, of ***events*** such as frame arrivals and timer expirations, and of ***conditions*** on the card configuration registers. The set of events generated and/or revealed by the card hardware, the set of actions coded in terms of pre-defined firmware modules, and the set of card registers whose settings can be tuned and verified, represent the device API that cannot be modified by the user.

The MAC program is coded into a transition table and loaded in a memory space deployed on the hardware. Starting from an initial (default) state, the MAC engine fetches the table entry corresponding to the state, and loops until a triggering event associated to that state occurs. It then evaluates the associated conditions on the configuration registers, and if this is the case, it triggers the associated action and register status updates (if any), executes the state transition, and fetches the new table entry for such destination state.

Since a MAC program is basically a list of labels specifying the events, actions and conditions associated to each state transition, by defining a common set of labels for the API (i.e. a machine language), the MAC program can be transported over data frames from one node to another. In [20] it has been shown that a basic version of DCF can be coded into 500 bytes only. By adding a simple header for controlling the loading and activation of the new state machine on the card, code mobility can be easily supported [20] in the so called MAClets (in analogy to the JAVA applets).

The MAC engine does not need to know to which MAC program a new fetched state belongs, so that a code switching is achieved by moving to a state in a different transition table and by updating the platform configuration registers (e.g. the operating channel, the transmission power, etc.). The definition of code switching transitions are logically independent of the MAC program definition. Therefore, rather than adding them to the MAC program, the architecture allows to program the switching transitions into a second-level state machine (meta state machine), whose states represent the MAC program under execution.

3. Architecture

In this section we present our solution for expressing and implementing high-level network policies in wireless local networks, in contrast to the current solutions based on

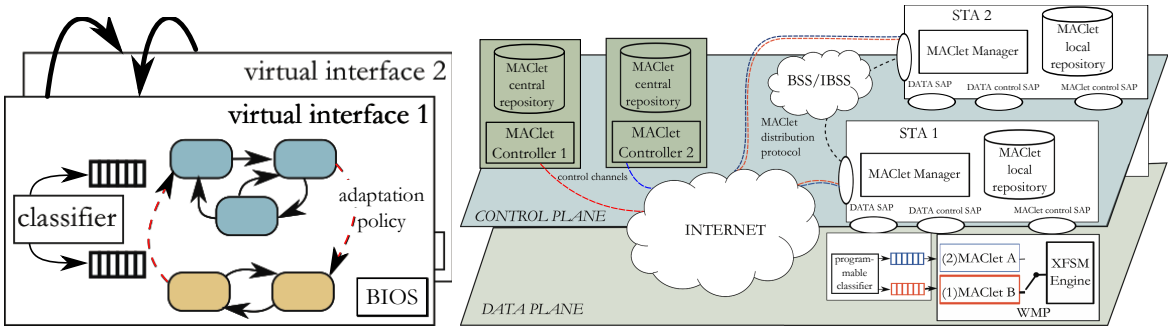


Figure 1: Architecture detail about Policy Control (a) and the Control and Data Planes (b).

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: i) it is based on the WMP API for collecting channel signals and statistics, ii) it exploits frame classifiers for managing multiple virtual interfaces, iii) 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.

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, multi-threading 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 1-(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.

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: (i) col-

lecting low-level information for estimating the network context; (ii) distributing and configuring MAC programs; (iii) ensuring against network inconsistencies in medium access rules. Figure 1-(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.

3.3 Control Messages and Procedures

Policies distribution among wireless nodes is performed in three (cooperating) ways: (i) the controller sends the MAClets to the MAClet Manager of each node via dedicated unicast control messages (specifically acknowledged); (ii) the controller sends MAClets in broadcast, by requiring that each MAClet Managers floods them into a a whole subnetwork; (iii) 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). Control messages are divided in Management, Action, Information, and Flow Control Messages, as summarized in table 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 2. 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

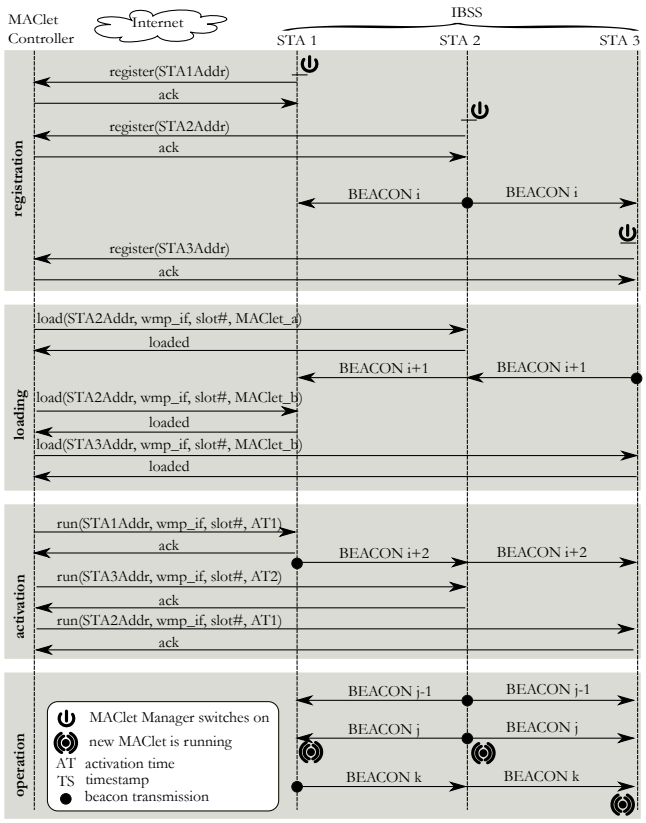


Figure 2: MAClet Distribution Protocol: message exchange and MAClet synchronization

MAClet Management Messages handle control channels Managers / Controllers	
register	(MM → 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 → MM), reply (MC ← 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 1: MAClet control messages

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.

4. Implementation

We implemented MAClet Manager and MAClet Controller as user-space applications, in agreement with the hardware-agnostic nature of MAClet code. The two applications build the control plane of a given network by embedding into UDP datagrams the protocol messages reported in Table 1. Control messages are transported into UDP packets, and can be unicast or broadcast. For example, MAClet action messages are broadcasted to all stations and filtered by receiving MAClet Managers accordingly to

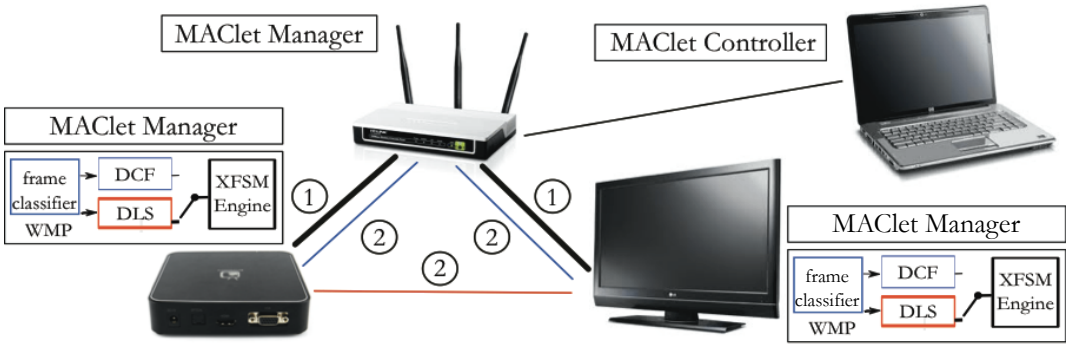


Figure 3: Use case: a TV box (left) delivers HD video to an Internet enabled TV while both communicates with Internet; a laptop (right) is connected to the internet via the AP.

the destination address. We reserved port number 1717 for the control plane protocol messages.

Multiple virtual networks can be defined over the same programmable nodes. On each virtual network, messages are exchanged according to the running MAC algorithm: for this reason a default algorithm is always virtualized, even if for very short intervals. This allows stations and their MAClet Controllers to load this default MAC, join the virtual operator(s), start a control plane session with some MAClet Manager and eventually switch to the right MAC. We use DCF as default.

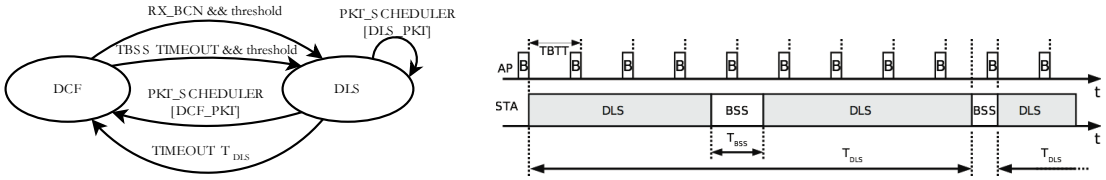
The session between the MAClet Manager and a MAClet Controller can be established upon a station joins a network. The communication between the MAClet Manager and the hardware is based on the WMP Control interface [20], able to inject MAC programs, activating a given program, collecting measurements from the corresponding interfaces, etc., according to the messages received by the MAClet Controller.

With respect to existing WMP engines, the firmware running in our testbed and deploying WMP on board of cheap off-the-shelf cards from Broadcom was carefully designed to allow quick MAClet switching.

At this early stage of development we have not introduced any mechanism for authenticating peers to each other and to control message integrity: we plan to implement a security sublayer by considering signatures and asymmetric cryptography which should not be problematic given that our software runs in user spaces with (usually) not constrained computational ability (e.g., smartphone or even better equipped devices).

5. Experimental Results

To prove the effectiveness of the available API and control messages for defining different adaptation policies, we tested our implemented architecture in a real scenario. Specifically, we considered a domestic infrastructure network, with an Access Point and a given number of associated stations in radio visibility, as shown in figure 3. The TV box, the Internet enabled TV, and the laptop usually require to be connected to the Internet (for downloading data or software updates). Additionally, the TV box may deliver a HD video to the smart TV. In such a case, according to the default DCF rules for infrastructure networks, the same data need to be transmitted from the TV box to the AP and from the AP to the smart TV, thus wasting bandwidth for the other nodes. Although current DCF extensions include the possibility to setup a *direct link*, i.e. a station to station data transfer without using the AP as relay, such a feature has to be



(a) (b)

Figure 4: Policy definition as a meta machine between two different threads (two virtual interfaces) (a) and DLS timing (b).

supported by the TV box and smart TV and activated by the user.

Suppose now that the ADSL provider, that is the owner of the home AP, wants to define a MAC adaptation policy according to which when two associated stations exchange a data volume higher than a given threshold, a direct link is activated between the two stations (for that data flow), while other traffic flows continue to be sent to the AP. The policy can be specified by programming a meta state machine as indicated in figure 4-(a), where two state machines (namely, the DCF and the direct link (DLS) one) are composed in a more complicated machine switching from one program to another according to the frame classifier output (DCF_PKT or DLS_PKT) and to the expiration of some timers. The two programs behave as different virtual interfaces and can even work on different channels. This is the reason of the switching timers, introduced for keeping the association to the AP even in absence of traffic flows for the AP. In our current implementation, T_{DLS} is 890 ms and T_{DCF} is 6 ms. The switching to DLS is enabled only when the packet counter of the station-to-station data flow overcomes a given threshold.

The policy is defined in the MAClet Controller and sent to the TV box and smart TV nodes by means of the MAClet distribution and synchronization protocol. Figure 3 shows the effect of this adaptation policy in terms of inter-node traffic flows (from scenario 1 to scenario 2): the TV box classifier routes video frames through the virtual interface that runs DLS (the red line), addressing them directly to the smart TV. The remaining traffic is sent to the AP, which then routes it towards the Internet (blue lines).

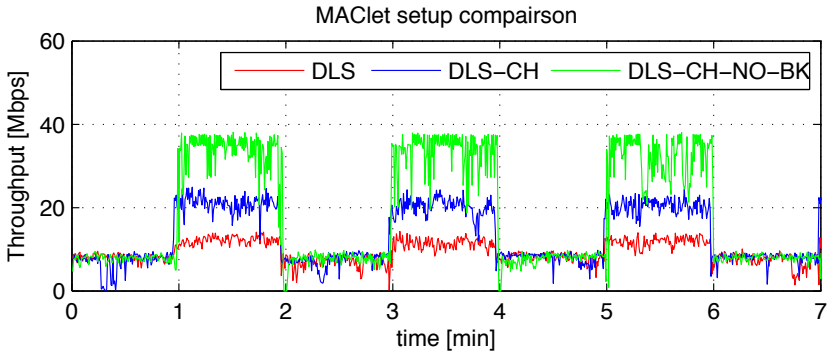


Figure 5: Throughput comparison under legacy DCF, DLS, and two versions of DLS++.

We set-up a testbed in our laboratory with two client stations acting as the TV box and smart TV (with a station-to-station traffic flow) and the AP equipped with a MAClet Controller and the envisioned policy. A third client was statically set to the AP channel with a legacy DCF protocol. We repeated the policy loading and activation test

periodically, by programming the AP to alternatively send (to the two programmable clients) the DCF-DLS MAClet and the legacy DCF MAClet at regular intervals of 1 minute. We tested three slightly different versions of the DCF-DLS MAClet: both the programs working on the same channel 6; the DLS program working on on channel 11; the DLS program working on the secondary channel 11 with a channel contention window set to 0. Figure 5 shows the throughput results of the client station sending saturated UDP traffic to the second client under the three settings (labeled, respectively, as DLS (direct link setup), DLS-CH (direct link with channel switch), DLS-CH-NO-BK (direct link on a different channel without backoff). Starting from legacy DCF, the clients switch to the different DCF-DLS configurations at 1, 3, and 5 minutes, and come back to standard DCF at 2, 4 and 6 minutes. From the figure it is evident that the customized direct-link access may bring dramatic improvements, especially when it is managed on a secondary channel without backoff (from about 12 Mbps of the normal DLS case to about 38 Mbps under the DLS without backoff).

6. Conclusions

This paper 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.

Acknowledgements

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

References

[1] J. Mitola III, G. Q. Maguire, “Cognitive radio: Making software radios more personal”, IEEE Personal Communications, vol. 6, no. 4, pp. 13–18, August 1999.

[2] V. Bose, D. Wetherall, J. Guttag, “Next century challenges: RadioActive networks”, ACM/IEEE MobiCom ’99, Seattle, USA, pp. 242–248.

[3] M. Neufeld, J. Fifield, C. Doerr, A. Sheth, D. Grunwald, “SoftMAC -Flexible Wireless Research Platform” HotNets, Nov. 2005.

[4] P. Djukic, P. Mohapatra, “Soft-TDMAC: A Software-Based 802.11 Overlay TDMA MAC with Microseconds Synchronization”, IEEE Trans. on Mobile Computing, Issue 99, April 2011.

[5] Open Network Foundation, `http://www.opennetworking.org`

[6] A. Campbell, H. De Meer, M. Kounavis, K. Miki, J. Vicente, D. Villela, "A survey of programmable networks", ACM SIGCOMM 1999, pp. 7 23.

[7] D. Wetherall, J. Guttag, D. Tennenhouse, "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols", IEEE Open Architectures and Network Programming, April 1998, pp. 117-129.

[8] R. K. Panta, I. Khalil, S. Bagchi, "Stream: Low Overhead Wireless Reprogramming for Sensor Networks", IEEE INFOCOM 2007, May 2007, Anchorage, pp. 928-936.

[9] M. Krasniewski, R. Panta, S. Bagchi, C.L. Yang, W. Chappell, "Energy-efficient on-demand reprogramming of large-scale sensor networks", ACM Trans. on Sensor Networks, February 2008, Vol. 2, pp. 1-38

[10] P. Levis and D. Culler, "Maté: a tiny virtual machine for sensor networks", 10th int. conf. on Arch. support for programming languages and operating systems, 2002, pp. 85-95

[11] 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.

[12] M.H. Lu, P. Steenkiste, and T. Chen, "Using commodity hardware platform to develop and evaluate CSMA protocols", ACM WiNTECH 2008, Sep. 2008, pp. 73-80.

[13] Open firmware for WiFi networks, http://www.ing.unibs.it/openfwwf

[14] B. Han, A. Schulman, F. Gringoli, N. Spring, B. Bhattacharjee, L. Nava, L. Ji, S. Lee, R. Miller, "Maranello: Practical Partial Packet Recovery for 802.11", USENIX NSDI10, Apr. 2010.

[15] The universal software radio peripheral, "http://www.ettus.com"

[16] Wireless Open-access Research Platform, "http://warp.rice.edu/trac"

[17] G. Nychis, T. Hottelier, Z. Yang, S. Seshan, P. Steenkiste, "Enabling MAC Protocol Implementations on Software-defined Radios", NSDI09, 2009.

[18] J. Ansari, X. Zhang, A. Achtzehn, M. Petrova, P. Mahonen, "Decomposable MAC Framework for Highly Flexible and Adaptable MAC Realizations", Proc. of IEEE DySPAN 2010, April 2010, pp.1-2.

[19] 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.

[20] G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, F. Gringoli, I. Tinnirello "MAClets: Active MAC Protocols over Hard-Coded Devices" ACM CoNEXT'12, pp. 229-240, 2012.

