One Size Hardly Fits All: Towards Context-Specific Wireless MAC Protocol Deployment

Giuseppe Bianchi
Universitá degli Studi di Roma - Tor Vergata, Italy
giuseppe.bianchi@uniroma2.it

Ilenia Tinnirello
Universitá degli Studi di Palermo, Italy
ilenia.tinnirello@tti.unipa.it

ABSTRACT

This paper casts recent accomplishments in the field of Wireless MAC programmability into the emerging Software Defined Networking perspective. We argue that an abstract (but formal) description of the MAC protocol logic in terms of extensible finite state machines appears a convenient and viable data-plane programming compromise for modeling and deploying realistic MAC protocol logics. Our approach is shown to comply with existing control frameworks, and entails the ability to dynamically change the MAC protocol operation based on context and scenario conditions; in essence, move from the traditional idea of "one-size-fits-all" MAC protocol stack to the innovative paradigm of opportunistically on-the-fly deployed context-specific MAC stacks. With the help of selected and currently developed use cases, we report on preliminary integration activities within the CREW federated wireless testbed, and its OMF experiment control framework.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless communication

Keywords

programmable MAC; WLAN 802.11, cognitive radio

1. INTRODUCTION

Originally introduced as "just" cable replacements, wireless networks have today dramatically expanded their scope. Market saturation, fierce competition, and traffic-revenue decoupling requires wireless operators to find creative technical means (other than pricing) to diversify their offers and get the most out of the current PHY technologies.

Flexibility and programmability of wireless devices appears crucial to foster wireless innovation [1]. Indeed, wireless service scenarios and application contexts evolve continuously and in an unpredictable way, and require significant and *fast* amendments of the underlying protocols. Customers should be given a personalized delivery service and quality of experience [2]. Wireless access performance should be matched to the nature of the application or service being used, and should be made able to smartly exploit opportunistically available spectrum and resources in dense environments [3]. And, finally, wireless protocols and solutions originally designed for general scenarios should be tailored (stretched?) so as to fit the specific needs of largely diverse (niche) contexts and deployments (industrial automation, domotics, military, emergency, machine to machine, etc).

In the attempt to make wireless flexibility and programmability viable, this work has a twofold goal.

First, we revisit recent work we carried out in [4, 5], where we introduced the notion of *Wireless MAC processor* (WMP). In a nutshell, the WMP design permits to decouple a set of "dumb" hard-coded, access primitives, from a third-party provided "smart" MAC protocol logic, according to which such primitives shall be executed. While in previous works we focused on the actual WPM core design [4], and on the technical extensions needed to run-time permit on-the-fly MAC protocol reconfiguration via "MAClets" [5], we here argue about the broader scope of WMP abstractions as data-plane interfaces for *Wireless-specific Software-defined Networking* (SDN) frameworks.

Second, in the attempt to identify practical control frameworks for WMP-enabled devices, we report on our integration experience within the federated experimental testbed developed in the frame of the CREW FP7 European project. Also via actually implemented and running use cases, we show how OMF controllers [6], so far meant for testbed's experiment control, can be easily extended to accommodate dynamic reconfiguration and deployment of context-specific MAC protocol, and can be thus envisioned as possible baseline controllers for wireless SDN scenarios.

2. MOTIVATION

Today, wireless technologies and protocols (in this paper we are specifically concerned with the Medium Access Control protocol, MAC) suffer of an extremely slow pace of innovation.

On one side, standardization organisations are slow and restrictive in adapting to the new evolving requirements. Slow because a standard, even when dealing with "soft" amendments at the MAC protocol logic level (versus, say, hardware changes in the physical layer primitives) may take years from the time it is chartered to the time it is finalized; restrictive because pushing standard bodies in launching a

WiNTECH'13, September 30 2013, Miami, Florida, USA

http://dx.doi.org/10.1145/2505469.2505482.

dedicated wireless standard amendment Task Group is all but easy, especially for small or medium enterprises, user communities or academic researchers.

On the other side, performance reasons require MAC protocols, and especially the low level MAC operation, to be implemented directly in the Network Interface Card (NIC). However, perhaps fearing commoditization, or perhaps simply not finding any compelling reason to take an alternative approach, most wireless manufacturers have pursued a closed products' design strategy. The aftermath is that today's commercial off the shelf wireless NICs remain unflexible, implement an *one-size-fits-all* standard MAC protocol, and expose very limited facilities (if any) to customize and/or adapt the channel access mechanisms to the possibly very specific and personalized context and service needs.

One-size hardly fits all

The wireless industry has so far standardized "protocols", including the MAC stack, buried once for all inside the NIC implementation, and great care has been obviously made in making the evolution of such protocols backward-compatible, via amendments and extension of a legacy protocol specification. At the opposite extreme, starting long time ago with cognitive [7] and active [8] wireless network visions, the wireless research and academic community has pushed forward dynamic reprogrammability of devices, so as to best fit unpredictable and dynamically mutating context situations, adapt to service demand variations, and smartly exploit temporarily unused radio spectrum.

In a fully programmable vision, a protocol stack should not be designed once for all, but the *most appropriate* protocol fitting the possibly *very specific* context at hands could be automatically downloaded upon need. Protocols would be simpler (for instance, why bothering with hidden terminals in contexts where there is none?), and backward compatibility would not be an issue anymore (all the stations in a same radio coverage could download and run the same stack).

However, it is unfortunate that the wireless reseach community, while pursuing such an idealized vision, did mostly focus on specific technology platforms rather than on open interfaces and means to formally describe a desired radio behavior. Indeed, many valuable programmable radio platforms have been developed in the course of the past years, including but not limiting to GNUradio [9], WARP [10], USRP [11], SORA [12], AirBlue [13], and including platforms specialized for MAC layer programmability, such as [14, 15, 16]. However, what appears largely under-explored[1] is the identification of *abstractions* and relevant programming languages which formally describe a desired wireless operation, and which can be deployed across multiple vendors' platforms (somewhat similarly to Java applets [1]), irrespective of their internal implementation (be it FPGA or DSP or a proprietary HW).

The need for abstractions (learning from SDN)

The impressive rise of Software-Defined Networking (SDN) in the wired domain has underlined the crucial role of vendor-independent behavioral abstractions for the data forwarding plane, originally pioneered by OpenFlow [17], but today not limited to it. SDN is not "just" about the separation of control and data planes, or "just" about the massive deployment of the OpenFlow protocol and its emerging extensions. Nor SDN should be confused with the complementary Network Function Virtualization (NFV [18]) trend of handling network functions as virtualized software-based services running on commodity hardware[2].

Rather, as crystal clarified in a talk by a leading SDN scientist[3], the SDN's paradigm-breaking vision that may obsolete the very idea of standards and protocols is the ability to control network functions and elements through viable abstractions "*consistent with the vendors' need for closed platforms*" (quoting [17]). These permit to dynamically provision, control, and even fully reprogram network nodes, using formal descriptions (e.g. an OpenFlow match/action table) of their forwarding behavior, provided by third party programmers via centralized remote controllers.

Arguably, the wireless enterprise community has anticipated, if not even inspired, the shift towards centralized controllers, but apparently has not been capable to recognize and harness the underlying SDN implications. Indeed, control/data plane separation and remote control of wireless access points was advocated as much as 10 years ago, and has lead IETF to charter, in 2005, the Control And Provisioning of Wireless Access Points (CAPWAP) protocol, later on standardized in RFC 5415. And the CAPWAP' "split MAC" idea of encapsulating MAC layer control frames and delivering them to a controller for centralized processing resembles the way OpenFlow delivers to the controller information about newly incoming flows.

What however the wireless community has somewhat neglected is the need to i) *devise viable programming abstractions for wireless functions*, and ii) show that they can be *handled with suitable extensions of already deployed network control frameworks*. Our proposals for addressing these two issues are presented in the next two sections, respectively.

3. WIRELESS MAC ABSTRACTION

Although [4, 5] do not mention SDN, in retrospective we believe that we therein identified a viable compromise between the ability to program a broad range of wireless MAC protocols, and the consistency with the vendors' need for closed platforms. We now briefly review the basic Wireless MAC Processor (WMP) concept and the relevant MAC programming abstraction, to the extent needed for understanding the new network-wide control and orchestration means described in the next section and integrated in the CREW testbed. The reader interested in technical details and implementation aspects (on a cheap commodity brand-name vendor's cards) is referred to the original works.

In designing a viable abstraction for platform independent wireless MAC protocols specification, the technical hurdle to

[1]Quoting Craig Partridge [1], "*One might imagine a lot of practical and theoretical work has been done on how to tell a radio how to behave and how a radio can describe its own behavior. However, rather stunningly, little work has targeted this problem*".

[2]Although with some stretch, NFV may be considered as the wired counterpart of the Software Defined Radio's ability to replace HW transceiver components with software programs. This analogy also helps to clarify why SDR and "wireless" SDN are very different concepts.

[3]Scott Shenker, The Future of Networking, and the Past of Protocols, Open Networking Summit 2011, Stanford, October 18-19, 2011; can be watched at http://www.youtube.com/watch?v=YHeyuD89n1Y

face is how to formally model the MAC protocol behavior using an high level language, and meanwhile support operations which may require a precision in the order of microseconds (e.g. schedule a frame transmission), and which cannot hence be outsourced to software programs running outside the NIC (e.g. in the driver). Our proposed abstraction is based on the following *decoupling* compromise:

- NIC cards do support an hard-coded (not modifiable by the MAC protocol programmer) *instruction set*, namely an Application Programming Interface (API) comprising of *actions*, *events* and internal parameters upon which *conditions* can be evaluated (see [19] for of our API's details)
- Third-party MAC programmers formally describe how actions are coordinated and triggered (by events and conditions) via eXtensible Finite State Machines (XFSM - [4]); in essence, the programmer can specify in a formal (executable) model, custom protocol *states*, state transitions and relevant triggering events and conditions, and actions invoked when state transitions occur and/or when a state is reached.
- To execute injected XFSM (suitably compiled into a byte-code-like language), the NIC further implements a generic XFSM processing engine, conceptually analogous to a Central Processing Unit (CPU) in ordinary computing systems, but technically differing in its operation, as its role is to fetch states, parse events, trigger state transitions, and invoke associated actions.

The above abstraction clearly decouples the role of the NIC manufacturer from that of the MAC programmer. Vendors remain in charge of providing HW signals and in bringing about innovation in the radio primitives (faster transmission technologies, advances in modulation and coding schemes, etc.); MAC programmers are free to define the protocol states and relevant transitions which orchestrate such primitives according to their desired MAC protocol logic. And dynamic MAC protocol reconfiguration becomes as easy as switching from a state machine to another.

4. USING OMF AS CONTROL FRAMEWORK

4.1 OMF Control Architecture

Several SDN concepts have been somewhat anticipated in the context of software frameworks for the configuration, execution and centralized control of wireless testbed experiments [20, 6]. Although conceived for benchmarking purposes, experiment controllers exploit some basic principles of software defined networking: i) build a global view of the network in terms of interactions between nodes; ii) provide a model of different network resources with different abstraction levels for configuring the desired behavior; iii) bind a set of resources to one central controller running a program which defines how to change resource properties and reconfigure nodes in reaction to events.

The first architecture developed according to these principles was the ORBIT [20] wireless testbed, whose main components, the cOntrol Management Framework (OMF) and Measurement Library (OML), are now deployed in several worldwide testbeds (from USA to Australia), including the European testbed CREW. In an OMF-enabled testbed, the

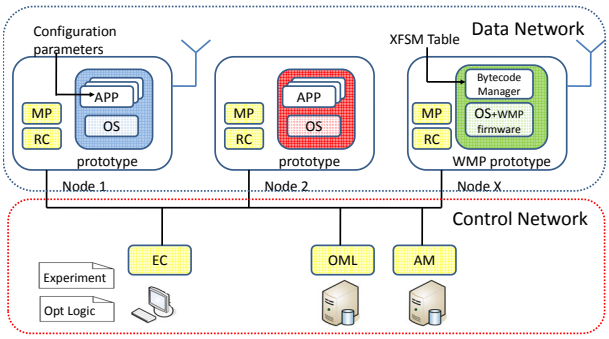

Figure 1: WMP integration in the OMF architecture as a specific node prototype.

global view of the network is given by the aggregation of different testbed resources in a *network topology* whose state is monitored by the aggregation manager (AM); the hardware and software network resources are abstracted in terms of nodes, interfaces and applications, that can be configured by means of a set of parameters called *properties* and combined in different *prototypes*; the experiments are managed by a central experiment controller (EC) able to send configuration commands to all the nodes and react to network events defined by the testbed users.

Fig. 1 shows the general architecture of an OMF-enabled testbed: the testbed nodes run a resource control process (RC) to interact with the EC, and one or more monitoring processes called measurement points (MPs) for collecting node statistics to be sent to the OML data base. Different messages can be exchanged between the resource control processes and the experiment controller: *loading messages*, for sending and installing the binary code of the application modules to the nodes; *call messages*, for launching the installed applications with specific configuration parameters; *event messages*, for signaling some events such as the successful installation of an application; *management messages*, for verifying the configuration and the status of the nodes. The controller sends the messages for driving the applications on the testbed nodes according to an experiment description file which allows to change the application parameters dynamically during the experiments. The experiment controller is also responsible for handling nodes' configuration consistency, synchronization problems, node state monitoring, etc., thus significantly simplifying the experiment description programmed by the users.

OMF testbeds have been used for experiments focusing on very different network aspects (from routing protocols, to content sharing applications and cooperative transmissions for ad-hoc networks) thanks to the application abstraction that allows to load on the testbed physical and virtual machines different network stacks and OSs, by linking interfaces and software modules (written in a generic programming language) with specific configuration parameters.

4.2 The WMP Prototype

Being the OMF programmability model based on composable software modules and parametric configuration interfaces, we considered the possibility to exploit the WMP abstractions for supporting a parametric definition of novel and dynamic MAC schemes in terms of coded XFSMs. To

```
defApplication('BM','bytecode-manager') do |app|
    app.description = 'Sothbound Interface to WMP'
    app.path = '/root/src/bytecode-manager'

    app.defProperty('listen', "enable XFSM remote
    injection", "-s ", {} )

    app.defProperty('connect', "send a XFSM to a remote
    note ", "-c ", {:type => :string, :default =>
    'localhost', :order => 1})

    app.defProperty('run', "activate the XFSM on a given
    position", "-a ", {:type => :integer,  :default =>
    1})

    app.defProperty('load', "inject a XFSM on a given
    card position", "-l", {:type => :string, :order =>
    1})

    app.defProperty('machine', "specify the bytecode
    text file", "-m ", {:type => :string, :order => 2})

    app.defProperty('dump', "outputs some card internal
    registers ", "-x ", {:type => :integer, :default =>
    1})
end
```

Figure 2: OML wrapper for the bytecode manager application.

this purpose, as depicted in figure 1, we defined a WMP prototype by integrating: i) the custom-made firmware developed for Broadcom wireless cards, ii) an interface application called bytecode manager, iii) a set of measurement points specifically designed for estimating the state of the wireless network.

Firmware. The firmware module allows to re-purpose the testbed wireless cards on which the prototype is loaded, by changing the card behavior from executing a standard 802.11 protocol to executing a generic programmable state machine. In comparison with the original version [4], the firmware has been slightly extended for enabling the possibility to collect low-level measurements to be exploited by the MPs. Two different extensions have been considered: the introduction of customized *statistic registers* that can be incremented by a relevant action at the occurrence of hardware events to be monitored; the possibility to filter the packets in the transmission and reception buffers by verifying a match condition in a two-bytes field in different frame positions.

Interface. For conveniently load a bytecode on the card or programming a switch to a new bytecode, our WMP release exploits an application called *bytecode manager*. To enable the exchange of messages with the EC, we developed a wrapper for using the bytecode manager as an OMF application. The wrapper allows to load the bytecode manager on the controlled node, invoke the WMP interface commands as different application properties, and track the output of the commands in the state of the controlled nodes. A simplified version of this wrapper is shown in figure 2. The wrapper, according to the SDN terminology, implements a *southbound interface* for configuring the node behaviors according to the injected XFSM.

Measurement Points. A set of auxiliary OMF applications have been developed for monitoring different network statistics on the controlled nodes. We developed a wrapper for the *tcpdump* packet analyzer whose properties allow to configure the packet filtering criteria, and a monitoring application of the WMP internal registers (called *WMPdump*)

```
Sensing Configuration
defGroup('N1', property.res1 ) do |g|
  g.addApplication('WMPdump') do |app|
    app.property('cw')
  end
end

Context Definition
defEvent(:HIGH_CW)  do |event|
  app_status =
group('N1').state("apps/app[@name='WMPdump']/io/out/line")
  if !app_status.nil?
    app_status.each do |element|
      event.fire if element>31
    end
  end
end

Reaction Mechanism
onEvent(:HIGH_CW) do |event|
   group('N1').stopApplication('WMPdump')
   group('N1').addApplication('bytecode-manager') do |app|
    app.setProperty('run', 2)
   end
   group('N1').startApplication('bytecode-manager')
end

Scenario
onEvent(:ALL_UP_AND_INSTALLED) do |event|
  group('N3').exec('iperf -s')
  wait 1
  group('N2').exec('iperf -c 192.168.0.3 -t 100 -i 1')
  group('N1').exec('iperf -c 192.168.0.3 -t 100 -i 1')
  wait 1
  group('N1').startApplications
  wait 30
  Experiment.done
end
```

Figure 3: An example of network control program written in OEDL.

whose properties allow to select the desired register and to convert the register value in decimal notation. Examples of parameters that can be collected by *WMPdump* are the number of correctly received or failed preambles, the number of CRC failurs, the contention window values.

4.3 Supporting a MAC cognitive cycle

To perform a functional analysis of the OMF primitives available for developing control applications, we consider a MAC cognitive cycle in which: i) the sensing phase is implemented by collecting different data by means of the monitoring applications deployed on the nodes; ii) the analysis and reasoning phases are performed at the EC by aggregating data and provide a network context estimate, iii) the adaptation phase is finally achieved by reprogramming, when needed, the MAC program on the controlled nodes.

Figure 3 shows a simple example of control applications, enlightening the functional mapping between the OEDL instructions and the different phases of the MAC cognitive cycle. The sensing phase is basically programmed by specifying the monitoring applications and parameters to be loaded on the network nodes (in our example, the *WMPdump* application will track the contention window value of node $N1$). The context estimation phase is programmed by defining customized events (i.e. contexts) to be identified when some specific conditions are met. For example, when the log of the monitoring application shows that node $N1$ is using a contention window higher than 31, an event is triggered for indicating an high contention state for that node. The reaction phase is programmed as a list of instructions (such as the switching to a different XFSM) to be executed after the

corresponding event is triggered. Finally, the figure shows the configuration of a network scenario when the OMF primitives are used for their native role, i.e. for configuring an experiment by specifying the traffic flows between the network nodes.

5. MAC ADAPTATION EXAMPLES

We deployed four WMP-enabled nodes in a CREW's testbed in Ghent (Wilab2). Nodes, called $Alix_i$, with $i = 1, \cdots, 4$ are equipped with the WMP prototype and MPs able to monitor high level performance figures and card registers. Node $Alix_3$ is configured to act as an Access Point because of its central physical position (with nodes $Alix_2$ and $Alix_4$, close each other, on the left and node $Alix_1$ on the right). For control purposes, nodes are wired to an independent Ethernet network, where the EC, AM and the OML data base also reside: we will show in the next section 5.3 how wired connectivity can be restricted to the AP only (as in real world conditions) and the remaining nodes can be reliably controlled via wireless, using wireless MAC virtualization.

Even if we rely on an experiment control framework, we stress that our usage of the EC should *not* be confused with the usual control of an actual experiment timeline (launching traffic flows, configuring wireless cards and MPs, etc). Rather, our scenario is that of an independent and autonomously evolving wireless network, but where dynamic MAC protocol adaptations are triggered by *context change estimates* according to a *centralized optimization logic*. Both such operations are implemented in OEDL, by defining different events and relevant actions.

5.1 Use case 1: performance adaptation

The example presented in this section (as well as the next one in section 5.2) is naïve, and the adaptation mechanism used can be obviously improved. Its purpose is however *not to* specifically propose an actual meaningful algorithm, but rather to showcase how an adaptation strategy can be *supported and orchestrated by the control framework*.

In WLANs, different channel conditions make such that stations may experience a largely different throughput despite using the same backoff settings. Fig. 4 shows that this phenomenon also occurs in our actual deployment. When node $Alix_1$, loaded with a greedy UDP data transfer (payload size of 1470 bytes) towards the AP, starts transmitting, it reaches maximum performance[4]. However, its throughput drops down to about 2.4 Mbps when node $Alix_2$ starts a similar UDP session, whereas $Alix_2$'s throughput is about 4.2 Mbps. As the initial session was not meaningfully affected by channel errors, this difference is likely caused by channel capture by node $Alix_2$, closer to the AP (upon collision, the AP is able to correctly demodulate the stronger $Alix_2$ signal).

Context gathering. This unfairness is detected by instructing the node MPs to periodically send throughput measurement samples to the EC. When a significant and stable difference in throughput is detected (a threshold percentage), the EC starts a closer investigation by sending a command to the *WMPdump* application for gathering the

[4]Considering overhead, the measured 6.7 Mbps with PHY rate of 11 Mbps is aligned with the expected saturation throughput for a single station in ideal channel conditions.

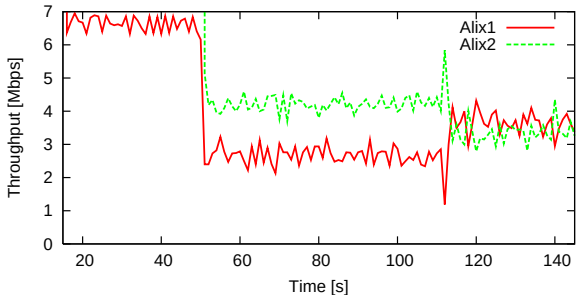

Figure 4: Unbalanced throughput results between $Alix_1$ and $Alix_2$ in case of capture effects.

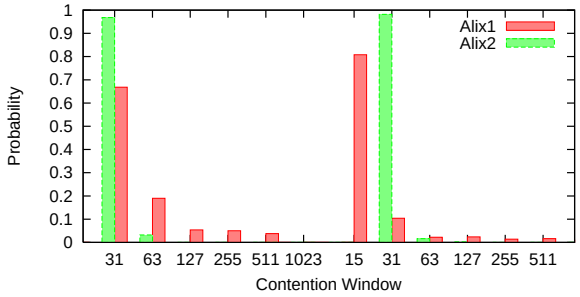

Figure 5: Distribution of the contention window values of $Alix_1$ and $Alix_2$ in case of capture effects.

contention windows distribution in 1000 transmission attempts (Fig. 5, left). The EC analyzes the data (an alternative cause could be the lower traffic generated by a station), notes the large amount of retransmissions by node $Alix_1$ versus the minimum contention window value used by $Alix_2$ in more than 95% of the cases, and diagnoses a *channel capture situation*, i.e., a new context.

MAC adaptation. The (trivial) implemented adaptation strategy consists in commanding the station with lower throughput to reduce the minimum contention window. Note that even if this is a simple change in parameters which could be actually supported by 802.11 drivers, in our proof of concept experiment we have actually implemented this as a *full change of the MAC protocol stack*, i.e. the EC delivers to $Alix_1$ a brand new XFSM (re)implementing the DCF protocol with $CW_{\min} = 15$, and sends an *activate* command. A more clever adaptation algorithm based on a more structural MAC stack change (e.g. from DCF to TDMA [21]) would of course use a different XFSM, but would be controlled and executed in the same manner as above. In Fig. 4, the switch command is sent at about time 110, as shown by the improved fairness (contention window statistics after the MAC protocol change are shown in Fig. 5, right).

5.2 Use case 2: topology adaptation

As a second use case, we integrated the Direct Link Setup (DLS) scenario described in [5] in the OMF control framework. We recall that DLS is an 802.11 protocol amendment devised to prevent the "triangular" routing and the relevant halve of capacity mandated by the original 802.11 standard, which forces two stations associated to a same AP to mutually communicate through the AP itself. DLS standardization has neither been easy nor fast (it was originally

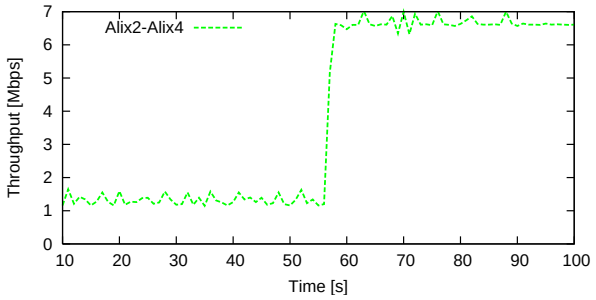

Figure 6: Throughput before/after DLS.

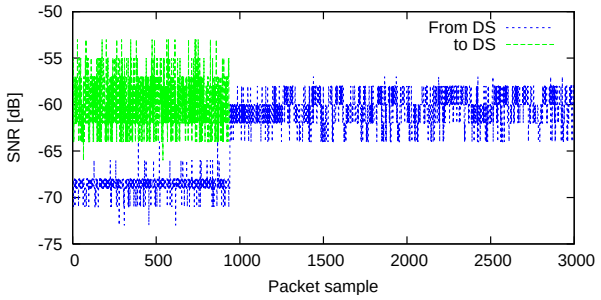

Figure 7: SNR samples monitored by node $Alix_4$

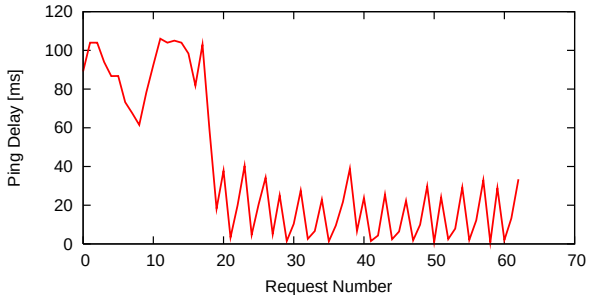

Figure 8: Ping delay measured from the AP to $Alix_2$ without/with virtual control interfaces.

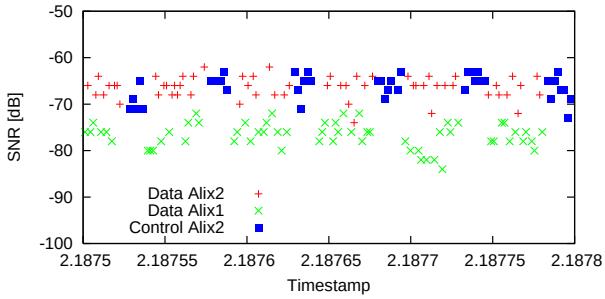

Figure 9: SNR samples monitored by the AP after activation of the virtual control interface.

introduced in 802.11e and further extended in the 802.11z-2010 amendment), as it is not nearly a trivial protocol since it has to deal with possible hidden terminal scenarios. In our deployment we set up a traffic flow from $Alix_2$ to $Alix_4$ which are in radio visibility. As shown in Fig. 6, the average throughput measured under standard DCF is lower than 1.5 Mbps, being the transmission rate in the AP-$Alix_4$ link set to 2 Mbps.

Context gathering. To understand the traffic scenario, the EC activates a *tcpdump* application on the AP. When an internal traffic flow between two mobile stations is revealed, the EC starts a closer analysis by also activating the *tcpdump* application on the destination node. The application is configured for filtering *from DS* and *to DS* frames and tracking the SNR values of each captured frame. An example of the resulting trace is shown in Fig. 7, where the *from DS* frames before the 1000-th sample are obviously sent by the AP, while the *to DS* ones are sent by $Alix_2$. The EC analyzes the data to understand if the two mobile nodes are visible or not, and detects that a direct link can be activated.

MAC adaptation. The direct link XFSM (available in the EC repository) is sent by the EC to both the mobile nodes with a different configuration parameter representing the MAC address of the peer node. After loading the program, the EC sends the switch commands to the new MAC protocol stack. According to our implementation, $Alix_2$ is now able to send frames to $Alix_4$ by pretending to be the AP. Fig. 6 shows the throughput improvements due to the protocol switch, while the SNR trace shows that the *from DS* frames are now sent by node $Alix_2$.

5.3 Use case 3: control network isolation

As a last example, we consider an adaptation problem in case of coexistence between the control and data networks, i.e. when the AP acts as a control message relay between the EC and the mobile nodes. The virtualization primitives of the WMP architecture can be exploited for isolating the control network, thus guaranteeing that control messages are correctly delivered to different nodes within some predictable time boundaries and that the measurement points can send back the monitored parameters by exploiting a guaranteed bandwidth. Indeed, the WMP nodes can load two different state machines on the controlled nodes (one for the data network and one for the control network) to be alternatively paused in a different portion of the beacon interval. Since all the nodes receive the AP beacon to keep the association, the time intervals in which they will transport data or control data will result synchronized.

Context gathering. To monitor the network capacity available for the control network, the EC (periodically or upon a specific signaling need) activates a ping application from the AP to the node to be controlled, whose delay results are sent back to the EC. Fig. 8 shows an example of delay measurements in case of greedy background traffic from nodes $Alix_1$ and $Alix_2$ to the AP. If the average delay or delay jitter is too high, the EC detects insufficient resources for the control network and triggers an adaptation.

MAC adaptation. If no virtual interface is configured on the nodes, the EC creates a dedicated interface for the control network and sends two DCF programs with a forced pause state, configured for being active in different portions of the beacon frame. If the virtual interface is already on, the EC sends a novel configuration parameter for (temporarily) increasing the portion of the beacon interval allocated to the control network. Fig. 8 shows the reduction of the ping delay when a third of a 50ms beacon interval is allocated to the virtual control interface (set up after 20 ping samples). Fig. 9 also visualizes the SNR values captures at the AP,

when a greedy transfer of control information is started from $Alix_2$ to the AP (with the previous background data traffic from $Alix_2$ and $Alix_4$). The figure clearly shows the time-based division between the control and data frames.

6. CONCLUSIONS

We believe that the ability to formally describe and instantly deploy custom wireless MAC stacks as eXtensible Finite State Machines, which orchestrate the execution of vendor's implemented radio primitives and signals, may obsolete existing one-size-fits-all standardized MAC protocols and pave the road towards multiple context-specific MAC protocols, optimized for niche scenarios and conditions. Backward compatibility is not anymore a necessary requirement (stations which need to interact would install and run a same specific protocol), but reduce to the need to agree on which actions, events and conditions (i.e., which API) should be supported by the vendors' cards.

This paper has further investigated control frameworks which may candidate to support and manage context-specific MAC protocol deployment and runtime adaptation. We specifically integrated our MAC programming abstractions into OMF, a control framework originally designed for the different purpose of controling the execution of testbed experiments, and showed how simple estimates of the network context can drive MAC protocol adaptations. A promising research direction consists in exploiting our MAC programming abstraction in the design of fully automated cognitive systems able to to smartly adapt (or even automatically reprogram) the channel access operation to mutating environments and traffic conditions.

Acknowledgments

This work has been carried out in the frame of the EU FP7-CREW project, contract number 258301. We thank Domenico Garlisi, Fabrizio Giuliano, Pierluigi Gallo and Francesco Gringoli for their support in the design and development of the WMP prototype development.

7. REFERENCES

[1] C. Partridge, "Realizing the future of wireless data communications," *Commun. of the ACM*, vol. 54, no. 9, pp. 62–68, 2011.

[2] M. El-Sayed, A. Mukhopadhyay, C. Urrutia-Valdés, and Z. J. Zhao, "Mobile data explosion: Monetizing the opportunity through dynamic policies and QoS pipes," *Bell Labs Tech. J.*, vol. 16(2), pp. 79–99, 2011.

[3] J. Zander and P. Mähönen, "Riding the data tsunami in the cloud: myths and challenges in future wireless access," *Commun. Magazine, IEEE*, vol. 51(3), pp. 145–151, 2013.

[4] I. Tinnirello, G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, and F. Gringoli, "Wireless MAC processors: Programming MAC protocols on commodity hardware," in *IEEE INFOCOM '12*, 2012, pp. 1269–1277.

[5] G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, F. Gringoli, and I. Tinnirello, "MAClets: active MAC protocols over hard-coded devices," in *8th ACM CoNext '12*, 2012, pp. 229–240.

[6] T. Rakotoarivelo, M. Ott, G. Jourjon, and I. Seskar, "Omf: a control and management framework for networking testbeds," *ACM SIGOPS Oper. Sys. Review*, vol. 43, no. 4, pp. 54–59, 2010.

[7] J. Mitola III and G. Q. Maguire Jr, "Cognitive radio: making software radios more personal," *Personal Commun., IEEE*, vol. 6, no. 4, pp. 13–18, 1999.

[8] V. Bose, D. Wetherall, and J. Guttag, "Next century challenges: Radioactive networks," in *5th ACM/IEEE Mobicom '99*, 1999, pp. 242–248.

[9] GNURadio open-source software development kit, http://gnuradio.org/redmine/projects/gnuradio/wiki.

[10] A. Khattab, J. Camp, C. Hunter, P. Murphy, A. Sabharwal, and E. W. Knightly, "Warp: a flexible platform for clean-slate wireless medium access protocol design," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 12, no. 1, pp. 56–58, 2008.

[11] USRP - the Universal Software Radio Peripheral, http://www.ettus.com.

[12] K. Tan, H. Liu, J. Zhang, Y. Zhang, J. Fang, and G. M. Voelker, "Sora: high-performance software radio using general-purpose multi-core processors," *Commun. of the ACM*, vol. 54(1), pp. 99–107, 2011.

[13] M. C. Ng, K. E. Fleming, M. Vutukuru, S. Gross, Arvind, and H. Balakrishnan, "Airblue: a system for cross-layer wireless protocol development," in *6th ACM/IEEE ANCS '10*, 2010, pp. 4:1–4:11.

[14] G. Nychis, T. Hottelier, Z. Yang, S. Seshan, and P. Steenkiste, "Enabling mac protocol implementations on software-defined radios," in *6th USENIX symp. on Networked systems design and implementation*, 2009, pp. 91–105.

[15] X. Zhang, J. Ansari, G. Yang, and P. Mahonen, "Trump: Supporting efficient realization of protocols for cognitive radio networks," in *IEEE DySPAN'11*, 2011, pp. 476–487.

[16] F. V. Gallego, J. Alonso-Zarate, C. Liss, and C. Verikoukis, "OpenMAC: a new reconfigurable experimental platform for energy-efficient medium access control protocols," *IET Science, Measurement & Technology*, vol. 6, no. 3, pp. 139–148, 2012.

[17] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Comp. Commun. Review*, vol. 38, no. 2, pp. 69–74, 2008.

[18] ETSI, "Network functions virtualisation - white paper," portal.etsi.org/NFV/NFV_White_Paper.pdf oct. 2012.

[19] Wireless MAC processor home page http://wmp.tti.unipa.it - documentation: https://github.com/ict-flavia/wireless-mac-processor/blob/master/doc/wmp-document.pdf.

[20] M. Ott, I. Seskar, R. Siraccusa, and M. Singh, "Orbit testbed software architecture: Supporting experiments as a service," in *1st IEEE Tridentcom*, 2005, pp. 136–145.

[21] I. Tinnirello and P. Gallo, "Supporting a Pseudo-TDMA Access Scheme in Mesh Wireless Networks," in *International Workshop on Wireless Access Flexibility, WiFlex 2013*, 2013.