Testbed Implementation of the Meta-MAC Protocol

Nathaniel Flick*, Domenico Garlisi†, Violet R. Syrotiuk*, Ilenia Tinnirello†,
* School of Computing, Informatics and Decision Systems, Arizona State University, Tempe, AZ, USA
† Dipartimento di Ingegneria Elettrica, Universitá di Palermo, Palermo, Italy

***Abstract*—The meta-MAC protocol is a systematic and automatic method to dynamically combine any set of existing MAC protocols into a single higher layer MAC protocol. We present a proof-of-concept implementation of the meta-MAC protocol by utilizing a programmable *wireless MAC processor* (WMP) on top of a commodity wireless card in combination with a host-level software module. The implementation allows us to combine, with certain constraints, a number of protocols each represented as an extended finite state machine. To illustrate the combination principle, we combine protocols of the same type but with varying parameters in a wireless mesh network. Specifically, we combine TDMA protocols with all possible slot assignments. We demonstrate that an implementation of the meta-MAC protocol over the WMP rapidly converges to non-conflicting TDMA slot assignments for the nodes.**

I. Introduction

In order to cope with changing conditions in the network, arising from changes in the traffic load or topology, most MAC protocols include some form of *adaptation*. More than a decade ago, a fundamentally different approach for adaptation was proposed in the meta-MAC framework [1]. It introduced a method to *systematically* and *automatically* combine any set of existing protocols into a single MAC protocol. Recently, advances in programmable radio platforms such as the *wireless MAC processor* (WMP) [2] and the *wireless open-access research platform* (WARP) [3], [4], have made the implementation of the meta-MAC protocol feasible.

Some examples of implicit combination of MAC protocols exist. For example, slotted p-persistence with dynamically adjusted retransmission probabilities p may be regarded as one. In a slotted p-persistent protocol, whenever there is a packet queued, the probability of transmission in a slot is a constant p, independently for each slot. If we dynamically change the value of p then we effectively combine p-persistent protocols that differ in their p values. Thus, in each slot we decide to use one of these component protocols, namely the one with the appropriate p. It is not an easy question which is the best way of adjusting the retransmission probabilities.

A more explicit combination of protocols is the idea of *protocol threading* [5]. Here, the basic idea is to interleave several different schedules (which may be different lengths and have different persistence) on a time sharing basis. In general, M schedules can be threaded together with each protocol used in every M^{th} slot. This was first used in threading *time spread multiple access* (TSMA) protocols [6], where the schedules allow collisions but are carefully designed to provide deterministic guarantees under certain conditions. When using TSMA protocols as components, the advantage is that the component protocols may be optimized for different densities of node topology. Then, the threaded protocol can handle situations without knowing in advance which one will occur.

Another explicit approach combines any schedule-based protocol with any contention-based protocol [7], in principle. The ADAPT protocol uses a simple TDMA protocol as the base protocol and combines it with CSMA/CA [8]. If the node "owning" a slot does not need to use it, other nodes can contend for it using CSMA/CA. Thus ADAPT can dynamically change its operation to reflect both the current load and node density.

The learning zero collision (L-ZC) protocol uses contention to allocate TDMA slot assignments [9]. In this way, L-ZC is effectively selecting between TDMA protocols with differing slot assignments based on network feedback from attempted transmissions.

The rest of this paper is organized as follows. §II describes the meta-MAC framework, and discusses implementation requirements on a real wireless card. In §III, we overview the *Wireless MAC Processor* (WMP) architecture meeting the meta-MAC requirements. §IV describes a firmware-level module for the execution of the selected protocol and the host-level module for protocol combination. The experimental setup on a wireless network testbed, along with the results of experimentation are found in §V, demonstrating a proof-of-concept implementation of the meta-MAC framework. Finally, we summarize and describe future work in §VI.

II. The Meta-MAC Protocol

As Fig. 1 shows, M MAC protocols $P_1, \ldots, P_M$ are combined at a given node; here M is not related to the number of nodes in the network, N. These component protocols can be arbitrary different protocols or they can be the same protocol but based on different parameters.

In order to simplify the presentation, time is divided into slots. Each protocol P_i runs locally and in each slot t produces a decision $D_{i,t}, 1 \leq i \leq M$, where $D_{i,t} = 1$ is interpreted as P_i transmits in slot t and $D_{i,t} = 0$ is interpreted as P_i does not transmit in slot t. A value $0 < D_{i,t} < 1$ is interpreted as a probability with which P_i transmits.

The meta-MAC protocol is an algorithm that runs locally at each node and combines the local decisions $D_{i,t}, 1 \leq i \leq M$, to produce a combined result D_t with the same interpretation as $D_{i,t}$. The final binary decision $\tilde{D}_t \in \{0, 1\}$ is derived from D_t by drawing a random binary value that takes the value 1 (transmit) with probability D_t and the value 0 (do not transmit) with probability $1 - D_t$.

978-1-4673-9955-5/16/$31.00 ©2016 IEEE

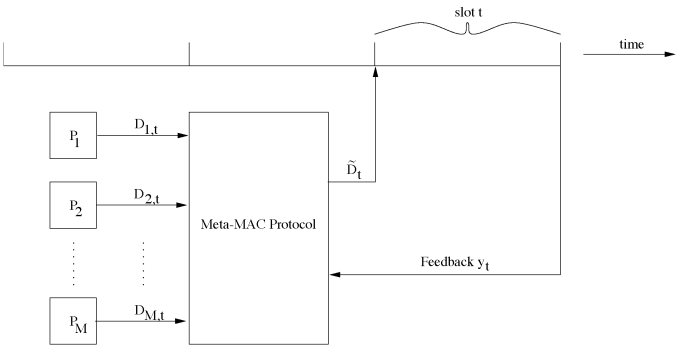

Fig. 1. Operation of the meta-MAC protocol.

The combined decision D_t is computed as a function of the weighted average of the $D_{i,t}$ values:

$$D_t = F\left(\frac{\sum_{i=1}^{M} w_{i,t} D_{i,t}}{\sum_{i=1}^{M} w_{i,t}}\right). \quad (1)$$

F is a function that grows linearly from 0 to 1 in an interval $[\frac{1}{2} - c, \frac{1}{2} + c]$ and is truncated to 0 and 1 before and after the interval, respectively [1]; c depends on another parameter η that controls how the weights are updated.

The meta-MAC protocol maintains the weights used in (1); $w_{i,t}$ is the weight of protocol P_i for slot t. At the end of each slot the weights are updated using the channel feedback. In a *ternary* feedback model, a node can determine whether a successful transmission occurred, a collision occurred, or the channel remained idle in a slot. We require information to be available for the meta-MAC protocol to conclude *at the end of the slot* whether the decision for the slot was correct.

For example, from the ternary feedback we can conclude the *correctness feedback*: If we decided to transmit and the transmission was successful then the decision was correct. However, if a collision occurred then the decision was incorrect. If there is a packet queued for transmission but we decide not to transmit there are two possibilities. If the channel was idle the decision was incorrect because the slot was wasted. However, if the channel was not idle then it was correct not to transmit as the channel was used by another node. If the queue was empty, then refraining from transmission was correct.

Given such correctness feedback, the weights are updated as follows. Let y_t denote the correctness feedback:

$$y_t = \begin{cases} 1 & \text{if the decision in slot } t \text{ was correct} \\ 0 & \text{if the decision in slot } t \text{ was incorrect} \end{cases}$$

Then the correct decision for slot t is $z_t = \tilde{D}_t y_t + (1-\tilde{D}_t)(1-y_t)$, where $\tilde{D}_t$ is the final binary decision as defined earlier. But we cannot set the decision for slot t to z_t because z_t only becomes known at the end of the slot. Using z_t, the weights are updated as $w_{i,t+1} = w_{i,t} \cdot \mathrm{e}^{-\eta|D_{i,t} - z_t|}$. The constant $\eta > 0$ controls how fast the weights change. This update rule can be interpreted as reflecting the "correctness history" of the component protocols.

A. Implementation Requirements

Implementing the meta-MAC protocol on a real wireless card it is not easy because of the following requirements.

Loading multiple programmable MAC components. Because of the strict timing constraints for accessing the wireless channel, MAC protocols are usually hard-coded in wireless cards or implemented in proprietary firmware. Assuming that it is possible to access and modify the firmware, such a modification is limited to programming a specific new MAC protocol that cannot be updated at run time without writing and loading new firmware. In some cases, non-critical MAC operations (*i.e.*, the upper-MAC) are implemented in the card driver, which in principle can be modified more easily. However, these operations are related to management operations, not to the lower-MAC transmission decisions.

Executing multiple MAC protocols in parallel. The MAC protocols defined in current standards are often based on a predefined combination of multiple operation modes. For example, in the Wi-Fi standard, a contention-based and a polling-based access scheme are both included in the access protocol. Several advanced features can be activated upon request and multiple contention parameters can be configured for different access categories contending simultaneously on the medium. However, this usual *combination* of multiple MAC components is not real parallel execution of different MAC components. Indeed, it is achieved by exploiting time-division (*i.e.*, by executing different access schemes in different time intervals) with an implicit organization of the channel time in contention-based or contention-free intervals, or explicit signalling between the stations for activating/deactivating a specific protocol variant. Only the EDCA protocol [10] represents a combination of independent MAC entities (one for each access category) which implements the same protocol with different contention parameters. However, the MAC entities can only configure the parameters of each component protocol, rather than dynamically associating a different protocol with each access category.

Combining transmission decisions. In current standards with multiple operation modes, the selection of a given component protocol is not performed on the basis of programmable logic based on the outcomes of previous transmissions. For example, in the case of EDCA, the contention parameters used by each access category are signalled by the *access point* (AP), while the combined transmission decision is established by the virtual collision mechanism and the priority of each access category, which cannot be set dynamically.

Processing per-slot channel feedback. Although many MAC protocols are able to gather per-slot channel feedback (*e.g.*, for decrementing or freezing the backoff counter or for updating the contention window), they cannot modify their operation logic as a function of this feedback. Moreover, this feedback cannot be exported to the driver at run time because of the unpredictable communication delays between the card and the host operating system.

III. A Multi-Protocol Execution Platform

To implement the meta-MAC protocol on common wireless cards involves running multiple (modifiable) protocol components, and switching from the decision of one protocol to another at run time without interrupting the card operations when loading and activating new firmware. Surprisingly, both the problems can be solved by working on the *wireless MAC processor* (WMP) which has been prototyped on top of a commercial wireless card. The prototype has been validated as a generic execution platform for MAC protocols including TDMA [11].

A. The Wireless MAC Processor Architecture

The WMP defines a new architecture for wireless interfaces, in which the medium access rules are not embedded into the firmware of the wireless interfaces, but can be programmed on the fly. For this purpose, the firmware implements the *MAC engine*, and a set of elementary actions and signal interrupts for interacting with the hardware transceiver.

The MAC engine executes MAC programs specified as extended finite state machines (XFSMs). This permits control of 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 hardware registers. The set of events generated by the transceiver, the set of actions coded in pre-defined firmware modules, and the set of hardware registers whose settings can be tuned and verified, represent the card API that cannot be modified by the user. The MAC program is coded as a transition table and loaded in a memory space 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 with that state occurs. It then evaluates the associated conditions on the configuration registers, and triggers the associated action and register status updates (if any), executes the state transition, and fetches the new table entry for the target state.

Since a MAC program is basically a list of labels specifying the events, actions, and conditions associated with each state transition, by defining a common set of labels for the API (*i.e.*, a machine language), the MAC program can be coded into compact bytecode, so that multiple programs can be preloaded onto the card. The MAC engine does not need to know to which MAC program a newly fetched state belongs. Code switching is achieved by simply moving from the current protocol state to a target state in a different transition table, with a latency of a few CPU clocks of the card. 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 exposes a control interface for loading new MAC protocols onto the card and for switching from one protocol to another.

Fig. 2 summarizes the components of the WMP architecture and how it has been used to implement the meta-MAC (see §III-B). The MAC engine is able to respond to the hardware interruptions and to trigger the hardware functionalities according to the logic defined in a selected bytecode.

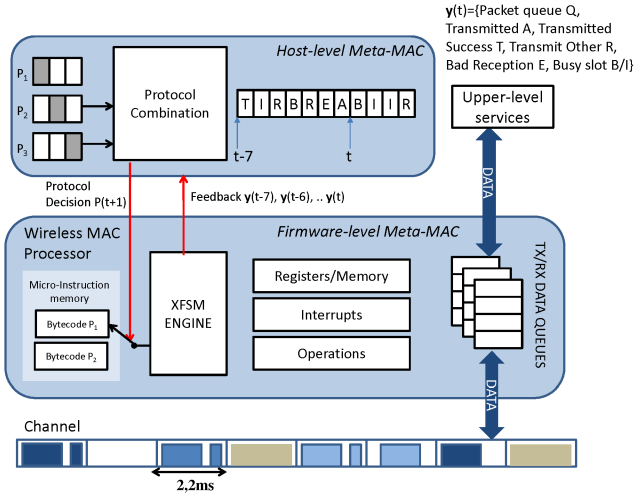

Fig. 2. WMP and meta-MAC architecture.

TABLE I
WMP API: SUPPORTED EVENTS, ACTIONS AND DATA FOR CONDITIONS.

Events	Actions	Data
CH_UP	rx_header()	channel
CW_DOWN	rx_msdu()	antenna
RX_PLCP_END	start_timer(reg,prm)	power
RX_MAC_HEAD_END	extract_bk(reg, prm)	txrx_on
RX_END	tx_start(prm)	backoff_slot
RX_ERROR	update_cw(reg, prm)	rx_checksum
QUEUE_OUT_UP	repor_to_host(prm)	busy_time
IFS_EXPIRED	start_ifs(prm)	backoff_value
TX_END	set(reg/var, var/prm)	bandwidth
TIMER_EXPIRED	get(reg/var, var/prm)	slot_time
	write(queue, field, var/prm)	+ *protocol*
	read(queue, field, var/prm)	*variables*
	incr(var)	+ *payload*
	hw_reset()	*fields*

B. Programming Component MAC Protocols

A version of the API originally proposed in [11] is summarized in Table I. In order to show the potential of the API for programming different component MAC protocols, Fig. 3 shows a simple state machine implementing a TDMA protocol. MAC program states are labeled, transitions are labeled by their triggering events, and conditions (when associated with a transition) are given in square brackets. The examples are limited to the management of frame transmissions (*i.e.*, acknowledgments and frame receptions are not included).

In Fig. 3, a simple TDMA program for accessing the channel access at regular time intervals is modeled with two states: a waiting state and a transmission state. From the waiting state, a transmission event is scheduled on a synchronization signal, given by the reception of a beacon frame. The event is scheduled after a time interval (*slot*) from the reception of the beacon header. When the timer expires, if the transmission queue is not empty, the transceiver is activated by calling the action *start_tx(queue)*. At the end of the frame transmission, a

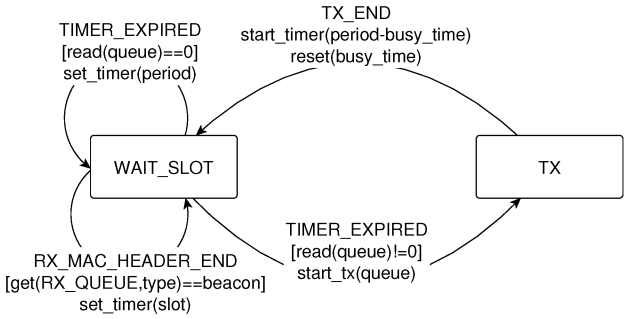

Fig. 3. An XFSM for the WMP API as a TDMA transmission protocol.

timer is set for the next transmission event by considering the difference between the protocol variable representing the inter-transmission *period* and the duration of the transmitter activity *busy_time*. When no frame is available for transmission, the same timer is set to the inter-transmission time.

IV. Meta-MAC Prototyping

Although the WMP platform allows the component MAC protocols to be programmed and loaded on a common execution platform, and to switch among components, there remain limitations for the implementation of the meta-MAC protocol. First, the MAC engine is single-threaded; *i.e.*, it is able to execute only one protocol component at a time. Multi-threaded generalizations are possible [12], *e.g.*, by allowing the MAC engine to work on a vector of states (one for each component protocol) and to solve hardware conflicts among multiple components. However, this generalization requires more complex firmware that cannot be supported by our WMP prototype. Second, the CPU on the card does not support floating-point operations required for the weight update rule and combining the protocol decisions.

To overcome these limitations, as shown in Fig. 2, we designed a meta-MAC architecture based on dividing functionality between (i) a firmware-level module for the execution of the selected protocol and the gathering of network feedback; and (ii) a host-level module for protocol combination, *i.e.*, emulating decisions and outcomes of non-running protocol components and selecting the firmware-level protocol.

A. Firmware-Level Protocol Execution and Network Feedback

The firmware-level module is based on the WMP prototype, which is able to execute a component MAC protocol selected from a list of pre-loaded MAC bytecodes. We restrict our attention to TDMA for the meta-MAC protocol validation. We use periodic beacon frames sent by the AP for synchronizing the stations and organizing consecutive channel slots into frames. We set the slot size to 2200 µs to accommodate a packet transmission and acknowledgment of 1500 bytes at 6 Mbps. The frame size is determined by a parameter in the running MAC bytecode.

The WMP firmware has also been programmed to save specific events in a channel trace that is periodically retrieved by the driver. In particular, at the end of each slot, the WMP firmware saves six binary feedback variables that collectively describe the state of the transmission queue, the protocol decision, the transmission outcome, and the slot state. The feedback variables and their semantics are described in Table II. In addition, the WMP firmware provides a 3-bit slot index counter, representing the current slot number modulo 8, and a value of a microsecond-precision counter which together are used by the host to determine the number of slots that have passed between consecutive reads. These records are stored in shared memory on the card and are accessed periodically by the driver.

TABLE II
Feedback variables and semantics.

Variable	Semantics
`packet_queued`	One or more packets queued for transmission.
`transmitted`	Transmission attempted.
`transmit_success`	ACK received for successful transmission.
`transmit_other`	Any data frame or ACK received.
`bad_reception`	Invalid data frame received.
`busy_slot`	More than $500us$ of channel activity in slot.

B. Host-Level Protocol Combination

The host-level module is a C program responsible for retrieving and processing the feedback provided by the card and selecting the MAC component protocol to be activated.

Of these responsibilities, retrieving the feedback from the card is the most challenging, because the host program must consistently read from the shared memory at least every seven slots (15.4 ms) to prevent newer feedback from overwriting older feedback that has not yet been read. For minimizing the probability of missing some information, we create one thread dedicated to reading from the WMP, and a processing thread responsible for everything else. However, there are no scheduling guarantees for the reading process and some feedback slots can be missed. In the case of missed feedback, we do not update the protocol weights for those slots; this is equivalent to assuming that there were no packets queued for transmission.

Once the feedback is retrieved, the host-level process has to emulate the decisions of each component protocol for each of the slots that has passed. This is critical because only one component protocol is actually running on the WMP platform, yet the decisions of all component protocols are required in order to generate the correctness feedback for the meta-MAC protocol. Thus, a software representation of each protocol, embedded in the meta-MAC host program, must be able to emulate the protocol's decision for each slot using the feedback provided. In the next section we describe how the six binary feedback variables provided by the WMP are sufficient for emulating the decisions of TDMA. Moreover, the feedback must be sufficient for calculating the correctness feedback for each protocol, given its decision in a particular slot. This second requirement is encompassed by the first. Using the feedback, the host program maintains weights for each of the component protocols according to the model described in

§II. Using these weights, the meta-MAC program then loads and activates MAC bytecode as necessary to keep the highest weighted component protocol running at all times.

Fig. 2 shows an example of three TDMA protocols with a frame of three slots. Protocol P_i transmits in slot i, $1 \leq i \leq 3$. In the example, the firmware-level MAC engine is executing protocol P_1. At time t, the feedback variables of the previous 7 slots are read by the host. The feedback is summarized as a label representing an aggregation of the flags (*e.g.*, A indicates that the transmitted flag is set to 1, while I indicates an idle slot in which the busy slot flag and the reception flags are set to 0). Assuming that the TDMA frame starts at the first feedback slot, the host-level meta-MAC can determine that protocol P_1 performed a correct decision at slot $t-7$, that protocol P_2 would have been successful in slot $t-6$ (which is idle), while protocol P_3 would have been incorrect in slot $t-5$ where frame transmission by another node has occurred.

C. Deviations from the Meta-MAC Protocol

The WMP architecture causes the implemented meta-MAC protocol to deviate from the model described in §II. In our implementation of the meta-MAC protocol, only the MAC bytecode associated with one component protocol (the highest weighted protocol) is running at any given time. Furthermore, because of communication delays and the fact that feedback is retrieved only every 7 slots, there may be a delay of potentially more than 15.4 ms between an event that ultimately causes the highest weighted protocol to change, and the time that the new bytecode is actually activated and running on the WMP. This may cause the decisions of our implementation, in situations where two or more of the highest weighted protocols have weights within an order of magnitude of each other, to differ from the decisions produced by the theoretical model. As the difference in normalized weight between the highest and the second highest weighted protocol approaches one, this discrepancy disappears and the decisions of our implementation asymptotically approach the decisions of the theoretical model.

Furthermore, the WMP architecture restricts the set of component protocols to those whose decisions can be emulated with only the feedback history provided by the WMP. TDMA protocols are possible using our implementation's six feedback variables, as they rely entirely on the slot numbering provided by the host program. Our implementation also supports slotted Aloha. With randomized protocols such as slotted Aloha, the host program needs only to calculate the probability that the protocol would have transmitted in a given slot.

V. Experimental Results

A. Testbed

We run our experiments in a testbed at the University of Palermo and in the wilab2.t testbed in Ghent, where 10 prototypes of the WMP architecture built on top of a commercial card by Broadcom are available.

Our implementation replaces the original card firmware with assembly code implementing the state machine execution engine, and mapping the previously described WMP programming interface into *actual* signals, operations, and registers of the card. The state of the hardware sub-systems (*e.g.*, the start of frame demodulation after a valid preamble) are automatically tracked by the internal card registers, enabling the logging of the meta-MAC relevant events. To support the upper-MAC operations and to interact with the meta-MAC C program, we use the *b43* soft-MAC driver, which adapts the Linux internal *mac80211* interface to the network card.

In all the experiments, the testbed has been configured in infrastructure mode, in order to exploit the beacon frames sent by the AP as a reference synchronization signal.

B. Optimizing TDMA Schedules

Simulations in [1] showed that component protocols can be arbitrary different protocols or they can be the same protocol with different parameters. In this second case, the meta-MAC can work as a protocol optimization framework. In particular, when each node runs multiple TDMA protocols differing in their slot assignment, the meta-MAC is able to reach a non-conflicting TDMA schedule in a distributed manner.

We reproduced this result in our experiments, by considering a simple topology of 4 nodes connected to the same AP. Each of these participants runs the meta-MAC protocol with four variants of TDMA; each variant works with a frame size of four but with slot assignments of 0, 1, 2, and 3, respectively. At the beginning of the trial, all protocol weights are equal and the TDMA slot 0 protocol is loaded and activated on each of the participant nodes. As with our other testing, we used a 6 Mbps bitrate and a 2200 µs slot length. The traffic is generated with the iperf program; the iperf server runs on the AP. We used a UDP stream with a bandwidth set to 10 Mpbs to ensure that the transmission queues of all nodes were saturated during the experiment. The duration of the experiment is 30 seconds.

The results of the experiment are shown in Figs. 4 and 5. Fig. 4 shows the slots on which transmissions were attempted and on which they succeeded, while Fig. 5 shows the protocol weights over the period during which the nodes were converging. It is clear that the nodes converged to a non-conflicting TDMA schedule. Specifically, it took about 19 slots for the nodes to converge on a non-conflicting schedule in this particular experiment. The learning constant η was set to 1.0 in this experiment; greater values do increase the speed with which the nodes initially converge. However, greater values increase the rate at which protocol weights both rise and fall, which counteract each other to an extent after the initial convergence, so the effects of η after the initial convergence are not as clear and require further research.

The slot assignments on which nodes alix10 and alix12 stabilized appear to differ between Fig. 4 and Fig. 5. This is because while the slots are synchronized among the nodes, the TDMA frames are not. Therefore, one node's local definition of slot 0 may not match that of another node. The slot assignments displayed in Fig. 4 are relative to the local definitions of node alix7, while the protocol weights displayed

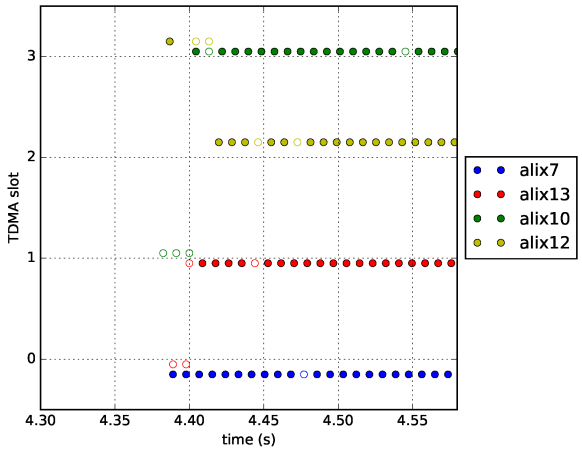

Fig. 4. Transmissions by TDMA slot assignment (offset within TDMA frame) over time for the period during which nodes converged on a non-conflicting schedule. Filled/hollow circles represent successful/unsuccessful transmissions. Nodes are offset from each other for clarity.

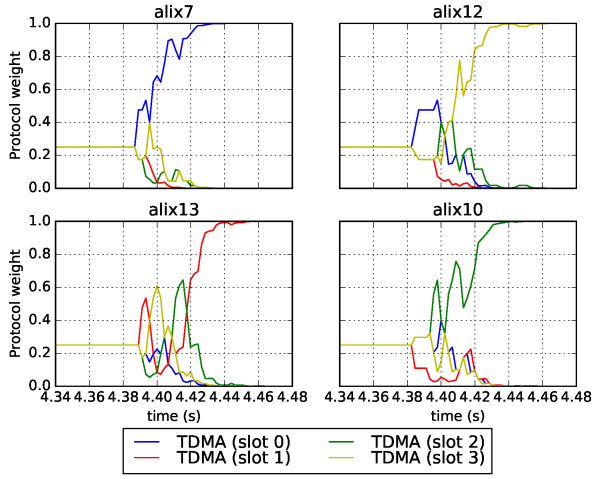

Fig. 5. Protocol weights over time for the period during which nodes converged on a non-conflicting schedule.

in Fig. 5 are local to each node. This is merely an issue of labeling, and does not affect the ability of the meta-MAC protocol to converge on a non-conflicting schedule.

In all experiments that we performed the nodes successfully converged to a non-conflicting schedule; all data was similar to that presented here. These experimental results demonstrate that our implementation is a successful proof-of-concept of the meta-MAC protocol.

VI. Conclusions and Future Work

This paper presented a proof-of-concept implementation of the meta-MAC protocol. Our implementation uses the Wireless MAC Processor to execute component MAC protocols on commodity hardware while a host-level program processes network feedback and selects the active protocol. We apply this implementation to TDMA protocol optimization and find that nodes converge to a non-conflicting slot assignment.

Our future work includes experimenting with different component protocols for the meta-MAC. In particular, transitioning between contention based and TDMA protocols in response to network load is an important result that has been established in simulation but remains to be demonstrated experimentally. We also intend to evaluate the effect of the learning parameter η on the convergence rate and stability of the meta-MAC.

It is also worth investigating the suitability of other programmable radio platforms for meta-MAC implementation. FPGA-based platforms such as the WARP may present solutions to constraints we faced in our implementation. An implementation of the WMP for the WARP platform has recently been developed, and this may prove to be a valuable starting point for this research.

In the long term, it may be worth investigating how the meta-MAC model can be generalized to non-slotted time. A completely general meta-MAC capable of combining *any* set of protocols would be a decisive leap forward in dynamic adaptation of MAC protocols.

Acknowledgements

This work has been supported in part by the EU H2020 WiSHFUL project, contract number 645274, and by the National Science Foundation, grant number 1421058.

References

[1] A. Faragó, A. D. Myers, V. R. Syrotiuk, and G. Záruba, "Meta-MAC protocols: Automatic combination of MAC protocols to optimize performance for unknown conditions," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 9, pp. 1670–1681, September 2000.

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

[3] "The wireless open-access research platform (WARP) project, Rice University," http://warp.rice.edu/trac/wiki/about.

[4] "The wireless open-access research platform (WARP), Mango Communications," http://mangocomm.com/.

[5] I. Chlamtac, A. Faragó, and H. Zhang, "Time-spread multiple-access (TSMA) protocols for multihop networks," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 804–812, 1997.

[6] I. Chlamtac and A. Faragó, "Making transmission schedules immune to topology changes in multi-hop packet radio networks," *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, pp. 23–29, 1994.

[7] I. Chlamtac, A. Faragó, A. Myers, V. R. Syrotiuk, and G. Záruba, "ADAPT: A dynamically self-adjusting media access control protocol for ad hoc networks," in *Global Telecommunications Conference (Globecom)*, vol. 1a, 1999, pp. 11–15.

[8] A. Colvin, "CSMA with collision avoidance," *Computer Communications*, vol. 6, no. 5, pp. 227–235, October 1983.

[9] M. Fang, D. Malone, K. R. Duffy, and D. J. Leith, "Decentralised learning macs for collision-free access in wlans," *Wireless Networks*, vol. 19, no. 1, pp. 83–98, 01 2013.

[10] G. Bianchi and I. Tinnirello, "Remarks on IEEE 802.11 DCF performance analysis," *IEEE Communications Letters*, vol. 9, no. 8, pp. 765–767, August 2005.

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

[12] Y. Grunenberger, I. Tinnirello, P. Gallo, E. Goma, and G. Bianchi, "Wireless card virtualization: From virtual NICs to virtual MAC machines," in *Future Network Mobile Summit (FutureNetw), 2012*, July 2012, pp. 1–10.