Report Number: WUCSE-2013-75

2013

Self-Adapting MAC Layer for Wireless Sensor Networks

Mo Sha, Meng Xu, Chenyang Lu, Linh T.X. Phan, Tae-Suk Kim, and Taerim Park

The integration of wireless sensors with mobile phones is gaining momentum as an enabling platform for numerous emerging applications. These mobile systems face dynamic environments where both application requirements and ambient wireless conditions change frequently. Despite the existence of many MAC protocols however, none can provide optimal performance along multiple dimensions, in particular when the conditions are frequently changing. Instead of pursuing a one-MAC-fit all approach we present a Self-Adapting MAC Layer (SAML) comprising (1) a Reconfigurable MAC Architecture (RMA) that can switch to different MAC protocols at run time and (2) a learning-based MAC Selection Engine that selects... **Read complete abstract on page 2.**

Self-Adapting MAC Layer for Wireless Sensor Networks

Mo Sha, Meng Xu, Chenyang Lu, Linh T.X. Phan, Tae-Suk Kim, and Taerim Park

Complete Abstract:

The integration of wireless sensors with mobile phones is gaining momentum as an enabling platform for numerous emerging applications. These mobile systems face dynamic environments where both application requirements and ambient wireless conditions change frequently. Despite the existence of many MAC protocols however, none can provide optimal performance along multiple dimensions, in particular when the conditions are frequently changing. Instead of pursuing a one-MAC-fit all approach we present a Self-Adapting MAC Layer (SAML) comprising (1) a Reconfigurable MAC Architecture (RMA) that can switch to different MAC protocols at run time and (2) a learning-based MAC Selection Engine that selects the protocol most suitable for the current condition and requirements. As the ambient conditions or application requirements change SAML dynamically switches MAC protocols to gain the desired performance. To the application SAML appears as a traditional MAC protocol and its benefits are realized without troubling the application with the underlying complexity. To test the system we implement SAML in TinyOS 2.x and realize three prototypes containing up to five MACs. We evaluate the system in controlled tests and real-world environments using a new gateway device that integrates a 802.15.4 radio with Android phones. Our experimental results show that SAML provides an efficient and reliable MAC switching, while adheres to the application specified requirements.

Department of Computer Science & Engineering

Washington University in St. Louis
School of Engineering & Applied Science

2013-75

Self-Adapting MAC Layer for Wireless Sensor Networks

Authors: Mo Sha, Rahav Dor, Gregory Hackmann, Chenyang Lu, Tae-Suk Kim, Taerim Park

Abstract: The integration of wireless sensors with mobile phones is gaining momentum as an enabling platform for numerous emerging applications. These mobile systems face dynamic environments where both application requirements and ambient wireless conditions change frequently. Despite the existence of many MAC protocols however, none can provide optimal performance along multiple dimensions, in particular when the conditions are frequently changing. Instead of pursuing a one-MAC-fit all approach we present a Self-Adapting MAC Layer (SAML) comprising (1) a Reconfigurable MAC Architecture (RMA) that can switch to different MAC protocols at run time and (2) a learning-based MAC Selection Engine that selects the protocol most suitable for the current condition and requirements. As the ambient conditions or application requirements change SAML dynamically switches MAC protocols to gain the desired performance. To the application SAML appears as a traditional MAC protocol and its benefits are realized without troubling the application with the underlying complexity. To test the system we implement SAML in TinyOS 2.x and realize three prototypes containing up to five MACs. We evaluate the system in controlled tests and real-world environments using a new gateway device that integrates a 802.15.4 radio with Android phones. Our experimental results show that SAML provides an efficient and reliable MAC switching, while adheres to the application specified requirements.

Type of Report: Other

Department of Computer Science & Engineering - Washington University in St. Louis
Campus Box 1045 - St. Louis, MO - 63130 - ph: (314) 935-6160

Self-Adapting MAC Layer for Wireless Sensor Networks

Mo Sha[1], Rahav Dor[1], Gregory Hackmann[1], Chenyang Lu[1], Tae-Suk Kim[2], Taerim Park[2]
[1]Department of Computer Science and Engineering, Washington University in St. Louis, USA
[2]Samsung Advanced Institute of Technology, Samsung Electronics, South Korea

***Abstract*—The integration of wireless sensors with mobile phones is gaining momentum as an enabling platform for numerous emerging applications. These mobile systems face dynamic environments where both application requirements and ambient wireless conditions change frequently. Despite the existence of many MAC protocols however, none can provide optimal performance along multiple dimensions, in particular when the conditions are frequently changing. Instead of pursuing a one-MAC-fit all approach we present a Self-Adapting MAC Layer (SAML) comprising (1) a Reconfigurable MAC Architecture (RMA) that can switch to different MAC protocols at run time and (2) a learning-based MAC Selection Engine that selects the protocol most suitable for the current condition and requirements. As the ambient conditions or application requirements change SAML dynamically switches MAC protocols to gain the desired performance. To the application SAML appears as a traditional MAC protocol and its benefits are realized without troubling the application with the underlying complexity. To test the system we implement SAML in TinyOS 2.x and realize three prototypes containing up to five MACs. We evaluate the system in controlled tests and real-world environments using a new gateway device that integrates a 802.15.4 radio with Android phones. Our experimental results show that SAML provides an efficient and reliable MAC switching, while adheres to the application specified requirements.**

I. Introduction

Research efforts in the last few years produced numerous MAC protocols for Wireless Sensor Networks (WSNs). Many of the MACs were optimized for low latency, high throughput, power consumption, or robustness to interference. However, none of the existing protocols deliver optimal performance in all desirable dimensions under varying environmental conditions. For instance, sender-initiated low-power listening (LPL) protocols (e.g., Box-MAC [24]) are efficient in clean environments, but suffer from high power consumption due to false wake ups in noisy environments [30]. Receiver-initiated MACs are more resilient to interference, but incur higher overhead at low data rates in a clean environment, since they periodically transmit probing packets. TDMA protocols are particularly good for high data rates applications by avoiding channel contentions, but CSMA/CA protocols incur less overhead and have shorter latency in low data rate.

A fixed MAC protocol prevents applications from meeting the demands of varying workloads, Quality of Service (QoS) requirements, or changing environmental conditions. This problem is exacerbated with the increased interest in connecting smart phones and wireless sensors placed on the user's body or in the surrounding environment. The fusion of a smart phone and a network of motes brings into existence opportunities for novel and exciting applications (e.g. fall detection, vital sign monitoring, assisted living, activity recognition, or fitness assessment) but WSNs, which used to be a stationary network of sensors, now faces new challenges. Which we articulate in the next few paragraphs.

First, the operating environment changes when the user moves through the world because of varying ambient interference. At times WSN will need to be able to deal with a highly noisy environment; at other times it may enjoy a clean environment. For example, Bluetooth devices, our own and our neighbors' Wi-Fi access points, and our microwaves – all generate high interference that interacts with WSN. This interference is unruly in homes and more orderly in office buildings [31]. Certainly a resilient MAC protocol may be required in noisy environments, while a different MAC be more efficient in a clean environments.

Second, the amount of data that a WSN carries is subject to spontaneous changes. For example, in a wireless health and assisted living applications the wireless sensors may produce low amount of data during some hours of the day, but sporadically, in response to a critical medical condition, require rapid transmission of large volume of data.

Third, different sensors have different traffic patterns and a WSN may turn ON or OFF any one of the sensors at any given time. For instance during stable periods, heart rate sensing may occur every minute and the data being sent is typically a single integer. But if the medical condition changes the application may activate the pulse sensor continuously, or decide to activate ECG sensors. The data rate can increase from less than 1 to 750 bytes per second [8].

Fourth, the application QoS requirements may also change. While it may be reasonable to lose a transmission every now and then during clinically-stable periods, it may not be acceptable during imminent clinical deterioration.

Having articulated the dynamic nature in which mobile WSNs may operate, the sporadic nature of some WSNs applications, and lacking the "Holy Grail" protocol, we assert that current applications, forced to use a single MAC protocol suffer from lack of adaptivity.

To fill this need we design a Self-Adapting MAC Layer (SAML) that makes available multiple MACs efficiently, and selects the protocol most suitable for the current conditions and requirements.

Our chief contributions are three-fold:

- RMA, a Reconfigurable MAC Architecture that supports dynamic switching among different MACs.

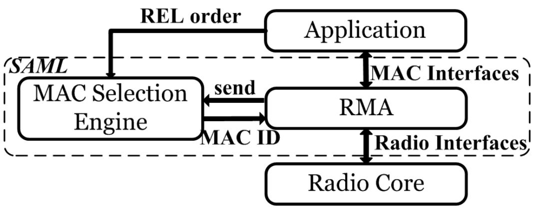


Fig. 1. Overview of System Architecture.

RMA holds multiple MACs without bloating its memory footprint due to its modular design based on shared components;

- MAC Selection Engine with a machine learnable model that optimizes along the dimensions of reliability, energy consumption, and latency and takes into account dynamic external interference;
- Implementation of SAML in TinyOS 2.x on the TelosB platform and a gateway device that we developed to integrate an Android smart phone with a 802.15.4 radio.

We validate the efficacy of SAML and the efficiency of its operation by measuring the memory footprint and the overhead of key operations. We also perform a four-day real-world case study in which we demonstrate that SAML provides efficient and reliable MAC switching, while adhering to the application specified requirements. Our case study shows that SAML saves 31.6% of energy compared with single MAC but still achieves the QoS reliability requirement of the application.

The rest of the paper is organized as follows. Section II presents the overview of our system architecture. Section III describes RMA architecture and Section IV shows how we realized RMA in TinyOS. Section V describes our MAC Selection Engine and Section VI presents experimental results. Section VII reviews related work and Section VIII concludes the paper.

II. Overview of System Architecture

In this section, we present the overview of the system architecture. Comparing with traditional architecture, we replace the MAC layer with the **RMA** and add the **MAC Selection Engine** as shown in Figure 1. RMA stores multiple MACs and supports switching between them at run time. The MAC Selection Engine is the component responsible for recommending the best MAC according to the specified QoS requirements, monitoring the dynamic ambient conditions, and automatically responding (without the need for the application to manage this process) to changes in the environment. Once a new MAC was determined by the engine, it will send the MAC ID to the RMA.

One of the primary design goals of SAML is to be transparent to its users. For this purpose SAML exposes a set of unified interfaces to the applications using it and to the lower radio layer. Application can treat SAML as a traditional single MAC entity and don't need to be manage any aspect of the MAC switches occurring in SAML. Five interfaces are available to applications: (1) Initialize and start/stop the MAC layer; (2) Control the radio CCA and backoff policies; (3) Send; (4) Receive; and (5) Specify the QoS requirements of the application. Interfaces (1)-(4) were shown to be enough to perform regular MAC operations [18] and (5) is provided for application to specify the QoS requirements as an 3-tuple that orders Reliability (R), Energy consumption (E), and Latency (L) in terms of their relative importance to the application. This is named the **REL order** of the application. The REL order can be updated by the application during run time to accommodate dynamic changes in its mode of operation. For example a duty-cycled clinical monitoring application which ranks energy consumption over reliability and reliability over latency will simply specify $E > R > L$ to achieve longer battery life. However, during a clinical deterioration period above REL should probably be changed to $R > L > E$ when the application becomes reliability-critical.

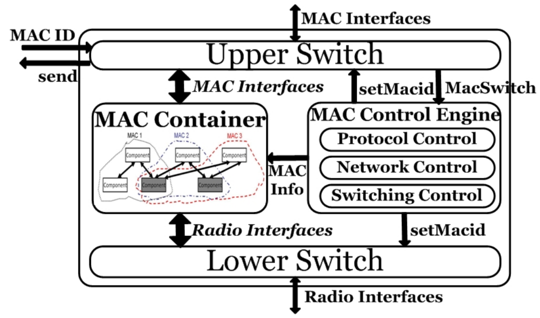


Fig. 2. RMA Architecture.

For the radio interfaces SAML adopts the same low-level interfaces suggested by [18]. These interfaces are platform independent, rather than specific to a particular radio or microprocessor, enabling portability between different hardware platforms.

We will describe the general design of the RMA in Section III and then present how we realize it in TinyOS in Section IV. We will show the design of MAC Selection Engine in Section V. We believe the architecture and design principles of RMA and MAC Selection Engine are applicable to other OS platforms. To enable the communication between an Android smart phone to communicate with a 802.15.4 radio, we have developed a Gateway device. The design of the Gateway can be found at Appendix I.

III. RMA Architecture

In this section, we present the RMA architecture. Our design addresses the following goals: (1) The architecture needs to present a unified set of interfaces to both its upper layer application users and lower layer radio developers. (2) RMA should reliably switch MACs while maintaining consistency between different nodes in the network. (3) The incorporation of multiple MACs should not significantly increase the memory footprint of the MAC layer. (4) Switching MACs should incur only a small run time overhead in term of both CPU cycles and communication bandwidth.

A. *Overview of the Architecture*

As Figure 2 shows RMA has four major modules. The **MAC Container** stores the MACs which are available at runtime. We leverage a component based architecture to reduce the memory footprint by allowing multiple MACs to share components (Section III-B). **Upper Switch** and **Lower Switch** provide a unified set of interfaces to application and lower radio core; abstracting away the details of the MAC layer

(Section III-C). **MAC Control Engine** controls the identification of the active MAC, manages the neighborhood table, and supervises protocol switching when it receives reconfiguration request from the MAC Selection Engine (Section III-D).

B. MAC Container

The MAC Container stores the MACs which are available at runtime. The design challenge we were facing is to enable multiple MACs without incurring a prohibitive increase in the code size. The naive approach would be to have the container encapsulate multiple monolithic MACs, resulting in a code size that is the sum of size of all individual MACs; but this would exhaust the onboard memory very quickly. An alternative approach is to distribute the new MAC wirelessly at run time during switching; but this would incur long latency, consume the network bandwidth, and spend precious energy on radio communication. The problem is exacerbated for mobile WSNs, such as a Body Sensor Network, since the MACs may need to change in a time scale of minutes due to the frequent changes in the environment or application needs.

MACs share many common functions and can be distilled to a set of reusable components [18]. To support variety of MACs with minimal penalty to code size our MAC Container is designed to hold these reusable components, from which MACs are built. The components inside the MAC Container can be (re)wired in different ways to construct various MACs. Our experimental results in Section VI-A show that this component based approach only moderately increases the code size when comparing RMA supporting multiple MACs even code using only a single MAC.

C. Switches

The Upper Switch and Lower Switch are two important design constructs of RMA. Their main purpose is to enable efficient (and possibly frequent) dynamic routing between components. All MACs within the MAC Container provide and use a uniform set of interfaces to the layer above and below the MAC layer. The switch is responsible for routing commands and events through the interfaces provided by the currently active MAC, using only a single variable to identify the active MAC. This variable is used as a *select* signal, determining which MAC is going to respond. This technique allows very quick protocol changes relative to many other alternatives such as dynamic loading of code functions, thread switching, and so on.

D. MAC Control Engine

The MAC Control Engine implements RMA core logic. It is designed to facilitate the identification of the active MAC, manage the network topology, and supervise the MAC changing process. It includes three major units:

1) Protocol Control: Protocol Control keeps track of the active MAC and makes sure all the components in the MAC Container are synchronized to the same MAC. The components are shared between different MACs and can be (re)wired in different ways to construct various MACs. The names of the components are maintained in a list in Protocol Control to address the components during MAC switching. When a MAC change occurs, Protocol Control treats the change process as a transaction. Either all of the updates occur and the protocol is changed, or the transaction will be rolled back.

2) Network Control: The Network Control unit manages the network topology. A node is specified by the application as a coordinator, which is responsible for nodes joining and leaving the network. In mobile WSNs, due to changing interference levels for example, this can occur frequently. The smart phone is usually a natural choice for the coordinator. To indicate the current MAC used by the network the coordinator disseminates an announcement packet, which includes the active MAC's identification (MAC ID) in each period T_1. If the current MAC uses beacons, RMA inserts a single byte into existing beacons instead of generating new ones to reduce communication overhead. For instance, RMA can use the time synchronization beacon of TDMA-based MACs.

Before joining a new node does not know which MAC is in use by the network and thus cannot join or talk with the rest of the network. For this purpose, we design a **baseline-MAC** which all nodes except the coordinator implement by default as part of RMA. The coordinator doesn't need this baseline-MAC and can run any MAC it wants when it boots, since all other nodes will synchronize their MAC to the coordinator. The baseline-MAC is designed to allow the initial formation of a network and to allow nodes to join or rejoin an already running network. When a sensor node boots, it first runs the baseline-MAC, which turns on the radio and overhears all packets. After it catches a protocol announcement packet from the coordinator it changes to this new protocol and sends a join-network request to the coordinator. When the coordinator receives the request it adds the new node to its neighborhood table and allocates resources for this node.

As discussed above new nodes, or nodes that lost connectivity to the network, may not know what MAC the network is currently running. Furthermore, at any given time the nodes need to know if they are connected to the network or not. We address these concerns by defining two time intervals: The coordinator needs to announce that it is present every T_1 seconds and nodes need to announce that they are connected every T_2 seconds. After joining the network, a regular node (non-coordinator) transmits a dummy packet to report alive each T_2 seconds, unless it happen to send data packets during this period. If the coordinator does not hear from a node (newly joined or one that was already in the network) within $5 \times T_2$ seconds it treats it as a node that left the network. It is removed from the neighboring table and all other resources assigned to it are deallocated. A regular node in the network changes back to the baseline-MAC and try to rejoin if it does not hear from the coordinator within $5 \times T_1$ seconds. This simple protocol maintains synchronization across the network, while accounting for the highly dynamic nature of mobile networks.

It is worth noting that we choose the values, 5, empirically, as they provided good balance between giving up too soon on a network node and allowing for dynamic nodes to join and leave as they wish. However the T_1 and T_2 factors do not need to be equal and other values can be used.

3) Switching Control: When a coordinator receives a MAC ID from its MAC Selection Engine, it verifies that the new MAC is different from current one and checks whether the requested MAC is stored in the MAC Container. If both conditions hold, the Switching Control unit notifies all sensor nodes in its neighborhood table and then performs the protocol change. It firstly refuses new packet issued by application, waits for all existing packets buffered in the MAC layer to be transmitted, and then shuts down the current MAC orderly. The variable holding the active MAC ID will be updated, and then new MAC will be started. If the new MAC has successfully started transmitting, then Switching Control unit returns success to the requesting application. Otherwise, it rolls back to the old MAC and retries the change. This process is allowed to repeat 30 times before the protocol change request is discarded. We choose the constant 30 empirically; Less attempts will consume less energy, but will result in higher chance of giving up.

IV. Realization of RMA in TinyOS

In this section, we describe how we realize RMA in TinyOS 2.1.1. While RMA has been implemented in TinyOS, we believe its design principles are applicable to other OS.

A. *MAC Container*

The MAC container stores various components from which MACs are built. Our impetus in creating the abstraction of the MAC Container is to minimize code size, which is essential given the limited resources on motes. Notably, for TinyOS the nesC compiler only creates one instance for each non-generic component, no matter how many times it is used by different code segments [1]. RMA reuses the components from the MAC Layer Architecture (MLA) [18], which distill common features of different MACs to a set of reusable components. Note however MLA was designed to facilitate the implementation of MACs at development time and does not support MAC layer reconfiguration at runtime. Our MAC Container encapsulates MLA components and adds new mechanisms to enable (re)wiring components such that they support runtime reconfigurations.

In TinyOS allowing components sharing raises the typical Fan-out issue of nesC, which is not a desirous effect for RMA. Here we briefly describe this issue and detailed explanation can be found in [19]. Fan-out: A single interface of a component (caller) is wired to two interfaces belonging to different components (callees). When the caller invokes a command on that interface, both callees will be invoked in an undefined order and may return different results.

When components are shared by different MACs they may be configured to perform different operations at different times. Only one MAC should run at a time and other MACs should stay inactive. Invoked by an command that was not meant for it, the code of inactive MACs may perform some unwanted operations, such as polling the channel, turning off the radio or starting an alarm, which may conflict with the active MAC. Our solution is adding a very lightweight **RMA Wrapper** that wraps each shared interface with a parameterized interface. In TinyOS, we use nesC parameterized wiring feature using the active MAC ID as the parameter. The code below shows the nesC code for a parameterized interface in such a RMA Wrapper.

```
module SharedComponent {
  //uses interface A;
  uses interface A[uint8_t id];
}
implementation {
  uint8_t currentMac;
  //call A.anycommand();
  call A.anycommand[currentMac]();
}
```

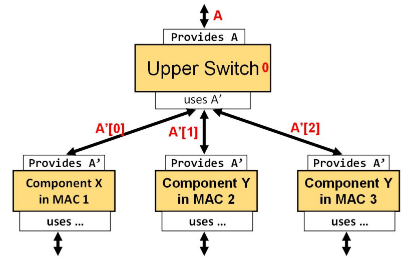


Fig. 3. Parameterized wiring based Switch design.

In each RMA Wrapper, a parameter is added on each interface declaration. A integer variable, "currentMac", indicates the active MAC ID. We use a local variable instead of a global variable to avoid potential race conditions. When a component calls a command through its parameterized interface, only the code of the current active MAC will be called. The same idea is applied when signaling an event. Each RMA Wrapper also needs to expose a `setMacid` interface for Switching Control unit to change the "currentMac" value. The MAC switching logic will be discussed in Section IV-C3.

B. *Switches*

Similar to the design of RMA Wrapper, we use nesC parameterized wiring to implement the Upper and Lower Switches of RMA in TinyOS. Figure 3 shows the design of Upper Switch which provides one single set of interfaces to the MAC Selection Engine, while using multiple set of interfaces provided by different MACs. Hence connecting many MACs to a single consumer. The active MAC ID is used to select between, or enable, only one of the MACs at runtime. The code below shows the nesC code for a parameterized interface that achieves this.

```
module UpperSwitch {
  provides interface A;
  uses interface A'[uint8_t id];
}
implementation {
 uint8_t currentMac;
 command void A'.command(){
    call A'.command[currentMac]();
  }
}
```

In the Upper Switch, a parameter is added on each interface that the Upper Switch provides. At runtime the variable, "currentMac", indicates the current active MAC. As explained before this parameterized interface declaration makes the upper layer application calls invoke only the code of the current active MAC. Upper Switch expose a `setMacid` interface for Switching Control unit to change the "currentMac" value. Upper Switch initializes the MAC switching through the `MacSwitch` interface when a new decision is generated by the MAC Selection Engine and its value is difference from "currentMac". The MAC switching logic will be discussed in Section IV-C3.

The same idea is applied in the design of Lower Switch, which allows many internal components of RMA to be connected to the single interface provided by the radio core. In the Lower Switch, a parameter is added on each interface that the Lower Switch uses to avoid fan-out problem when signaling an event.

C. MAC Control Engine

1) Protocol Control: An integer variable named "currentMac" is maintained in all RMA Wrappers and Switches, indicating the active MAC ID. The Protocol Control unit uses an array to store the list of RMA Wrappers and it uses this list to treat a MAC switch as a transaction. Either all wrappers and switches will change their "currentMac" value to indicate the new protocol, or none of them will. In TinyOS, this is achieved using an atomic block.

2) Network Control: Network Control, when running on a coordinator node, maintains a neighborhood table to keep track of the nodes in the network [1]. We reserve the first byte in payload packets for broadcasting the MAC ID. Nodes that did not join the network run the baseline-MAC and use this field to recognize the current MAC. RMA requires each MAC to provide a method for new nodes to join the network. For instance, we observed that some MACs cannot deal with network topology changes since they do not support methods for new nodes to join or leave the network. Pure TDMA implemented by [18] is such a protocol. It initially accounts for the nodes in the network and divides a fixed time window to slots for the number of nodes currently in the network. With all the time slots allocated to nodes, no new node can join. We accounted for that in our own version of pure TDMA by reserving one of the slots to an arbitrary new node. When more than one node is trying to join the network they will compete for this slot using a CSMA-style technique.

3) Switching Control: When the coordinator decides to switch MACs it sends out the switching command to every node in its neighborhood, and then reconfigures its MAC layer to the new MAC. Similarly, every regular node switches to the new MAC after receiving the switching command. Since the delivery of a packet with switching command is critical for protocol synchronization across the network, the Switching Control unit on the coordinator node uses a single hop unicast with Automatic Repeat reQuest (ARQ) to notify the new MAC to each node in the network[1]. The coordinator treats a node as a one that left the network and will release its resources after 30 failed attempts.

Switching Control exposes to the Upper Switch the `MacSwitch` interface. When the `changemac` command is invoked by the Upper Switch, Switching Control unit calls the `stop()` command of the `SplitControl` interface to shut down the current MAC whose MAC ID is "currentMac" value. It then changes "currentMac" values through the `setMacid` interface in each RMA Wrapper based on the list maintained in Protocol Control as well as the values in the Switches. After this is completed it invokes the `start()` command to turn on the new MAC, whose MAC ID is the new "currentMac" value.

The `stop()` command of the `SplitControl` interface is critical for RMA for performing MAC change. `stop()` command does not attract enough attentions by MAC developers since it is not commonly used in a single MAC environment. When we reviewed the original implementations of MACs we found out that the `start()` command, which is used to start the MAC, is carefully implemented; but often the `stop()` command is vacuous or not carefully implemented. Sometimes developers forget to stop the timer or alarm which was started in the protocol's `start()` command or during the protocol execution. This may cause unreliable operation in a multi MAC environment. Therefore, we emphasize that all logic units started during protocol initialization and execution must be stopped within the implementation block of the `stop()` command.

D. Implementations

We obtained the source code of MLA, which works with TinyOS 2.1.1 from [22]. The source code includes the implementation of BoX-MAC [24] and pure TDMA. We also obtained the implementation of RI-MAC [33] from its authors [2]. As an exercise, we implement an adaptive TDMA based on pure TDMA (this protocol allows reconfiguration of TDMA frames at run time) and a ZigBee MAC based on the standard [37].

The naming we used in the TinyOS RMA are as follows: We implemented `SwitchUpC` component as Upper Switch and `SwitchLowC` component as Lower Switch. The `MacC` component is the MAC Container and all the functionality in the MAC Control Engine are fulfilled by `ProtocolsControllerC`. The baseline-MAC is implemented in the `BaseLineC` component.

We have built three prototypes using our TinyOS implementation of RMA: a CSMA/TDMA prototype, a SI/RI-MAC prototype, and a 5-MAC prototype. The CSMA/TDMA prototype includes a CSMA/CA MAC (BoX-MAC) and a TDMA MAC (pure TDMA). The SI/RI-MAC prototype includes a sender-initiated MAC (BoX-MAC) and a receiver-initiated protocol (RI-MAC). Lastly the 5-MAC prototype includes all five MACs (BoX-MAC, pure TDMA, RI-MAC, adaptive TDMA, and ZigBee MAC).

As a summary for future researchers we summarize the procedure of creating a RMA prototype:

1) Implement the MACs you desire using the reusable components provided by MLA; Make sure each MAC provides a method for adding new nodes to the network;
2) Identify the components which are shared by these MACs and wrap each of them with a RMA Wrapper (discussed in Section IV-A).
3) Assign each MAC a unique integer as its MAC ID;
4) Add the names of these shared components into the RMA Wrapper list (discussed in Section IV-C1);
5) Wire the interfaces exposed by the individual MACs to the parameterized interfaces exposed by `SwitchUpC` component and `SwitchLowC` component (discussed in Section IV-B) using MAC ID as the parameter.

V. MAC Selection Engine

In this section we present the design of the MAC Selection Engine.

[1]In principle, RMA can be extended to support multi-hop networks. Dissemination protocols (e.g. [20]) can be used to broadcast the active MAC ID or a switching command in a multi-hop network. Mechanisms to ensure agreement on the MAC among sensors within the network will need to be employed [4].

[2]The authors' implementation is based on TinyOS 2.0.2, which we port to TinyOS 2.1.1

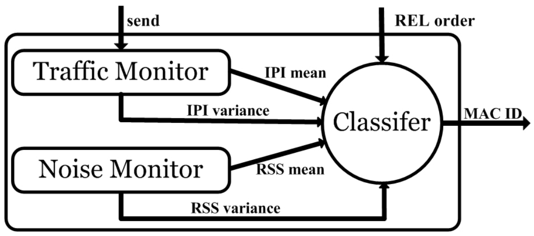


Fig. 4. MAC Selection Engine.

A. Overview of the Engine

The MAC Selection Engine decides based on a machine learnable model which is the best MAC protocol for given application QoS requirements (a REL order), current traffic pattern, and ambient interference levels. As shown in Figure 4, the engine includes three major modules: The **Traffic Monitor** keeps track of the application traffic pattern by snooping the send commands called by the application and calculating the mean and variance of Inter-packet Interval (IPI) in a sliding window of 100 seconds. The **Noise Monitor** measures the external interference level in the environment by calculating the mean and variance of the Received Signal Strength (RSS) in a sliding window of 10 seconds. We choose RSS as an indicator estimating the interference level since recent study showed that 802.15.4 sensors can effectively detect external interferers by polling Received Signal Strength Indicator [12]. The **Classifier** determines the best MAC according to the current application specified REL order and the values emitted from the Traffic and Noise Monitors. The Classifier then issues its decision (MAC ID) as a request to the RMA to switch MACs. The detailed design and implementation of the classifier is described in Section V-B.

We emphasize that only the network coordinator node needs to run the engine and the coordinator's RMA is responsible for propagating the MAC protocol switching across the network (See Section III-D3). The other nodes need to have only the RMA module.

B. Classifier

We choose to use a decision tree as the classifier for a number of reasons: There are only a handful of MACs that can practically be available in any container; and each protocol is generally good at some characteristics (e.g. most efficient at low data rates) and is not so good at others. So often the classifier will only have limited, discrete choices to make. When we consider the limited amount of memory available on sensors, decision trees are compact to represent in a data structure. And from computational point of view a decision tree will consume marginal amount of resources and will be fast at run-time.

To gather training data for the model we run experiments that vary the features we were interested in, while recording the operating characteristics and the MAC protocol in use (the class). We determined the features by noting that some protocols are good at high data rates, while others conserve energy and are good at low rates; and some protocols stay reliable at high interference levels, while others do not. With that in mind the characteristics we use for classification are:

1) Application QoS requirements

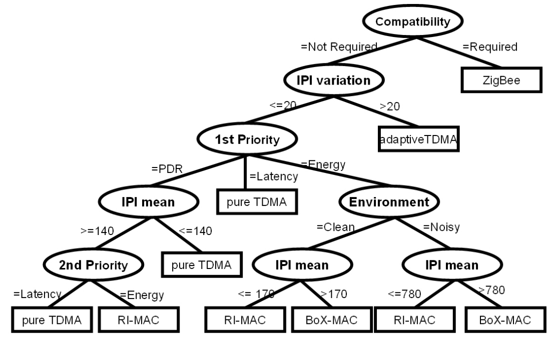


Fig. 5. Decision tree.

2) Traffic pattern
3) Ambient interference level

To represent these characteristics we selected the following features: (1) Application specified REL order; (2) Mean and variance of the Inter-packet Interval (IPI) within a 100 seconds sliding window; and (3) Mean and variance of the RSSI within a 10 seconds sliding window. The protocol being used is recorded and denotes the class.

The Energy that each MAC protocol consumes under certain operating conditions is computed by the time duration when radio is on/off. We define Latency is the time interval between a packet being sent by one sensor and the time it is received at another. We use Packet Delivery Rate (PDR) to represent Reliability.

We have ran 224 sets of experiments, varying IPI and ambient interference by changing the distance from a pair 802.11 devices with controlled traffic. We measure each combination of operating characteristics multiple times, and collected the operating results of various MACs in terms of the features described above. In total we collected 4624 training examples. Each example is calculated within a sliding window of 10 seconds. We used Weka [35] to learn from the training data and built a decision tree using the C4.5 [27] algorithm. The resulting model (tree) is shown in Figure 5.

Referring to Figure 5 before decisions are made by the decision tree, if compatibility with other ZigBee devices is required, then at the root node the system instructs to use the ZigBee MAC which was designed for compatibility across different platforms. Otherwise, if the IPI variance is higher than 20 ms^2, the classifier selects adaptive TDMA protocol. This is an apt choice that the model learned, since adaptive TDMA allows reconfiguration of TDMA frames to accommodate aperiodic traffic. The next level of the tree takes into account the application's highest REL order (the application's 1st priority) and provide a 3-way split. If the split was on Reliability, the next level of the tree (IPI mean) takes into account the mean IPI. If the Traffic Monitor observes small mean IPI (lower than 140 ms) the system is instructed to use pure TDMA. Otherwise the application 2nd priority is taken into account. If the split was on Energy consumption, the next level of the tree (Environment) takes into account the noise level.

To learn the mode and test its performance measures we used the 10-fold cross validation of Weka. The model obtains a true positive rate of 95.6% and a false positive rate of 7.1%. These performance measures demonstrate that our features can effectively select the best MAC for the specified application's

	ROM (bytes)	RAM (bytes)
BoX-MAC	25308	1114
pure TDMA	25362	1202
RI-MAC	25132	1268
adaptive TDMA	25418	1126
ZigBee MAC	27168	1272
RMA CSMA/TDMA	28016	1254
RMA SI/RI-MAC	27752	1896
RMA 5-MAC	29990	1968

TABLE I. ROM AND RAM USAGE FOR EACH RMA PROTOTYPE OR SINGLE MAC.

REL order, traffic pattern, and current ambient interference level.

VI. EVALUATION

We validate the efficacy of SAML and the efficiency of its operation in numerous ways. We start by measuring memory footprint of the three prototypes discussed in Section IV-D. In these prototypes RMA hosted between 2 to 5 MAC protocols in its container. This is a typical range and should provide a good representation of code size. We then measure the overhead of key RMA operations such as new node joining the network and MAC switching. We intentionally disable the MAC Selection Engine and manually inject the MAC IDs to support the experiments presented in Section VI-B. Finally, we enable the MAC Selection Engine and perform a real-world case study in which we demonstrate the effectiveness and benefits of dynamic MAC switching in terms of reliability and energy consumption.

A. Memory Footprint

RMA's design had to balance two conflicting goals. On one hand we want RMA to host as many MAC protocols as any application may require to optimize its performance along on some future, and possible unknown, dimension. On the other hand, artlessly, more protocols will increase the OS code size. We addressed this conflict by breaking MAC protocols to reusable components, resulting code size that is a concave down function of the number of protocols.

In Table I we compare the ROM and RAM usage as reported by the TinyOS tool-chain for five single MACs as well as three RMA prototypes discussed earlier.

Comparing the RMA CSMA/TDMA prototype containing two MACs (BoX-MAC and pure TDMA), with the one containing only BoX-MAC, we observe that the RMA consumes only 2708 additional bytes of ROM and 140 additional bytes of RAM [3]. This is only 10.7% ROM and 12.6% RAM increase from a single MAC. Compared to pure TDMA only, RMA prototype consumes 2654 additional bytes of ROM (10.5%) and 52 additional bytes of RAM (4.3%).

RMA SI/RI-MAC prototype with two MACs (BoX-MAC and RI-MAC) adds only 2444 bytes of ROM (9.7%) and 782 bytes of RAM (70.2%) compared to BoX-MAC. RMA adds 2620 bytes of ROM (10.4%) and 628 bytes of RAM (49.5%) compared to RI-MAC. We are not concerned with the high RAM percentage because in terms of percentages from the mote resources the percent increases are similar to the ones reported in the next paragraph (less than 8% in the worse case for all cases we have studied so far).

[3] For reference the MSP430F1611 MCU used by the TelosB and Tmote Sky motes provides 48 Kilobytes of ROM and 10 Kilobytes of RAM.

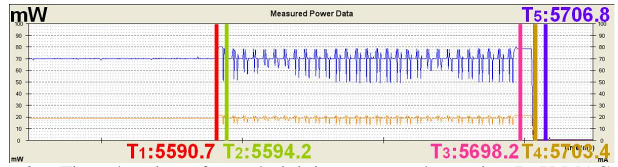


Fig. 6. Time duration of a node joining a network running BoX-MAC.

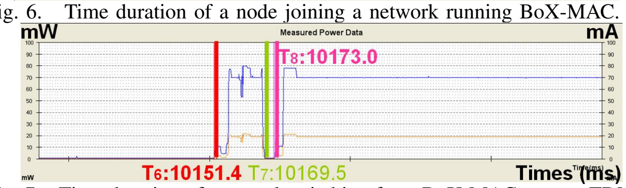


Fig. 7. Time duration of protocol switching from BoX-MAC to pure TDMA.

Comparing the 5-MAC RMA with the ZigBee MAC which consumes the most memory out of all single MACs shows that the RMA version consumes 2822 additional bytes of ROM (10.4%) and 696 additional bytes of RAM (54.7%). This increase represents a 5.7 percent increases in the ROM and 6.8% increase in RAM of the mote's memory resources.

In all cases we clearly see that RMA is highly effective in avoiding memory bloat. This conclusion holds even when supporting a large and diverse set of MACs, which makes RMA practical to deploy on memory constrained sensors.

B. Micro-benchmark Experiments

We evaluate the latency and power consumption when new nodes join the network and switch between MACs. We measure the consumption using a power meter from Monsoon Solutions [23] whose probes are connected to the sensor voltage pins. For this experiment, we use a Samsung Galaxy Note with a gateway board (discussed in Appendix I) that works as the coordinator. The coordinator is initialized to run BoX-MAC with a wakeup interval of 150 ms and broadcasts the MAC ID every 5 seconds. A mote running the baseline protocol was added to the network at $T_0 = 0$. After 10 seconds, we issue a command through the phone requesting a switch from BoX-MAC to pure TDMA. The pure TDMA frame is configured to include 20 time slots with 10 ms for each slot. The 10th time slot is configured to allow unknown node to perform CSMA-based random access. A second mote running the baseline protocol was added to the network after 5 seconds.

Figure 6 shows the time duration of the first mote joining this network (running BoX-MAC). After booting, the new mote begins to run the baseline-MAC, snooping the channel continuously. At T_1 = 5590.7 ms, the node receives a broadcasted packet with MAC ID of the current protocol sent by the coordinator. At T_2 = 5594.2 ms, the node starts sending requests to join the network until it receives an acknowledgement to do so at T_3 = 5698.2 ms. This request process takes $T_3 - T_2 = 104$ ms and consumes $7.29mJ$ of energy (70.13 mW of power on average) [4]. At T_4 = 5703.4 ms, the new mote turns off the radio, switches the MAC from baseline-MAC to BoX-MAC and then performs low power listening. The switching process $T_5 - T_4$ takes 3.4 ms and consumes $2.86\mu J$ of energy (0.84 mW of power on average).

Figure 7 shows the case where the mote receives a command requesting it to switch from BoX-MAC to pure TDMA. At T_6 = 10151.4 ms, the mote wakes up and receives the

[4] This duration depends on BoX-MAC wakeup interval.

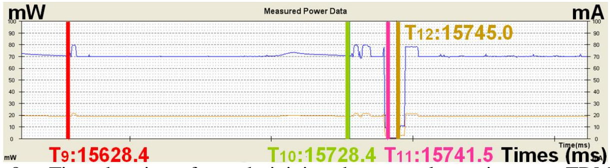


Fig. 8. Time duration of a node joining the network running pure TDMA.

switching command. Then the mote sends back an acknowledgement at T_7 = 10169.5 ms, turns off the radio, and starts switching the protocol. The radio is turned back on at T_8 = 10173.0, now waiting for a TDMA time synchronization beacon. The switching process $T_8 - T_7$ takes 3.5 ms and consumes $2.87\mu J$ of energy (0.82 mW of power on average).

Figure 8 shows the time duration of a second mote, also running baseline-MAC, joining the network running pure TDMA. At T_9 = 15628.4 ms, the mote receives a broadcasted packet with MAC ID. With the frame information stored in the beacon the node realizes that the 10th time slot can be used for new node to randomly access the channel. At T_{10} = 15728.4 ms the node performs a CCA check and then transmits a request to join the network. After receiving an acknowledgement the node turns off the radio at T_{11} = 15741.5 ms and switches from the baseline-MAC to pure TDMA. It then waits for a TDMA beacon at T_{12} = 15745.0 ms.

For each RMA prototype we randomly generate 100 MAC switching commands through the phone with random intervals ranging from 5 seconds to 10 minutes and obtain a result of 100% of MAC switching success rate. From the power meter traces we observed that the switching process takes about 3.5 ms and consumes about $2.94\mu J$ of energy on average. This short transiting time is achieved by the RMA's fast MAC switching design and this low switching power can be explained by the radio chip being off during the switching. These results demonstrate the efficiency of RMA in terms of controlling nodes as they join the network or switch between different MACs.

C. Case Study

To illustrate the potential benefits of dynamic MAC switching, we perform an empirical case study emulating a wireless health scenario. In our hypothetical scenario the application periodically (twice per second) samples the person's pulse and oxygen saturation in the blood by using a wireless pulse oximeter. The application starts to collect a 1-hour continuous ECG streaming when an abnormality in these vital signs is detected. In this emulation we do not perform the sensing because the actual values are irrelevant to the evaluation of SAML. Instead, a corresponding (equivalent to the real application) rate of packet generation is maintained by generating packets following the traffic pattern of 1 packet/500 ms for pulse and oxygen saturation sampling suggested by [6] and adopt a packet rate as high as 1 packet/50 ms with a payload of 15 bytes to accommodate the 500 Hz 12 bits ECG sampling recommended by [8]. In our wireless health scenario, we set (E $>$ R $>$ L) as the REL order during the clinically (periodic pulse and oxygenation sampling) normal period and set (R $>$ L $>$ E) during the clinically abnormal (ECG streaming) period.

For our case study a Ph.D. student volunteered to wear a TelosB mote on his wrist, a second TelosB mote on his chest [5], and the TelosB gateway in his pocket. The volunteer carefully repeated the same daily schedule over four days to provide similar environments as we collected data for the different protocols.

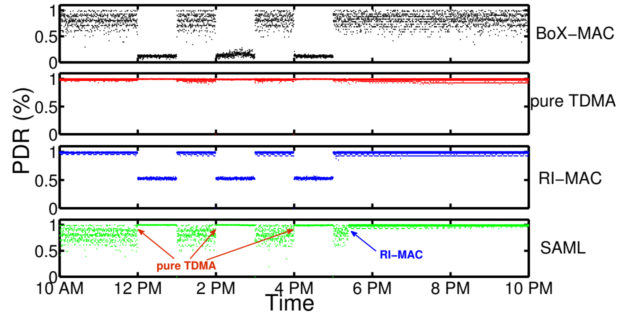


Fig. 9. PDR of BoX-MAC, pure TDMA, RI-MAC, and SAML during 10 hours.

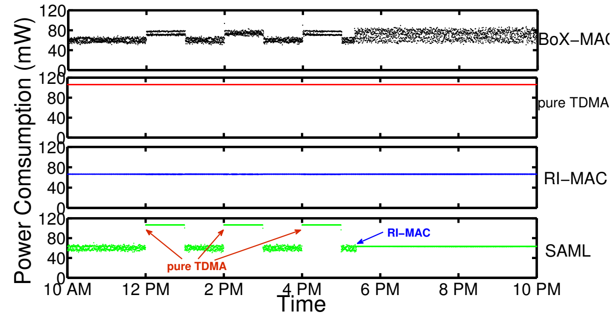


Fig. 10. Power consumption of BoX-MAC, pure TDMA, RI-MAC, and SAML during 10 hours.

To compare the performance between single MACs and SAML the volunteer wore the motes for 12 hours (from 10:00 to 22:00) during four consecutive weekdays and went about his regular daily activities. The activities were repeated at about the same time each day. The motes run our SAML with 5 MACs on the first day, and BoX-MAC, pure TDMA, RI-MAC, one on each of the respective day afterwards. During these days we intentionally emulate three clinically abnormal events to trigger the ECG streaming at 2 hours, 4 hours, and 6 hours after the experiment started each day. Since our volunteer was active and mobile throughout this experiment we could not measure the power consumption directly. Thus, we instrumented the radio stack, measured the radio ON time T_{on}, and use the actual duty cycle to estimate the energy consumption rather than measure it directly by a power meter. We use the equation $U_{on} * I_{on} * T_{on} + U_{off} * I_{off} * T_{off}$ to estimate the power consumption in each cycle (parameters U_{on}, I_{on}, U_{off}, and I_{off} are from CC2420 data sheet).

Figure 9 and Figure 10 show the raw data of reliability in term of Packet Delivery Rate (PDR) and power consumption during the 10 hours case study. From the figures, we can see that SAML uses BoX-MAC when performing the pulse sampling during the first 2 hours. When an abnormal-vital event is triggered at 12pm, SAML switches to pure TDMA to accommodate the high data rate generated by ECG streaming.

[5]These two places are where the typical commercial heart rate and ECG sensors would be placed (e.g. Polar Rate Monitor [26], Shimmer ECG mote [32]).

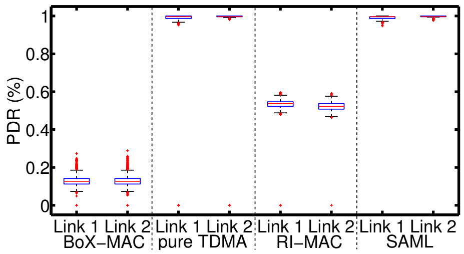


Fig. 11. Box-plot comparison on PDR of two links during three ECG streaming between BoX-MAC, pure TDMA, RI-MAC, and SAML. Central mark in box indicates median; bottom and top of box represent the 25th percentile (q_1) and 75th percentile (q_2); crosses indicate outliers ($x > q_2 + 1.5 \cdot (q_2 - q_1)$ or $x < q_1 - 1.5 \cdot (q_2 - q_1)$); whiskers indicate range excluding outliers.

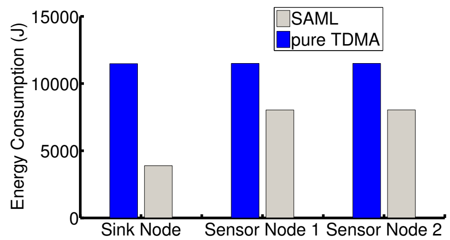


Fig. 12. Comparison on total energy consumption over three nodes during vital signs sampling between pure TDMA and SAML.

SAML switches back to BoX-MAC at 1pm when ECG streaming is stopped. Same pattern repeated at both 2pm and 4pm when two other abnormal-vital event are triggered. Around 5:30pm, SAML switches to RI-MAC when it detects a noisy environment at volunteer's home.

Figure 11 shows box-plot of PDR between two links during the three ECG streaming. The plot compares BoX-MAC, pure TDMA, RI-MAC, and SAML in terms of maintaining the link reliability. Only pure TDMA and SAML achieved a PDR higher than 99.6% and met the application reliability requirement (the application specified that as the top priority during ECG streaming). BoX-MAC and RI-MAC fails to provide reliable link and yielded a median PDR of 12.7% and 52.9% respectively. Pure TDMA however, suffers from very high energy consumption due to its fixed time frame and periodic beaconing. From Figure 10 we see that during clinically normal period (when data rate is low) pure TDMA (106.4 mW) consumes 77.3% and 60.5% more power than BoX-MAC (60.0 mW) and RI-MAC (66.3mW), respectively. A system with a single MAC, in this case pure TDMA, would not be able to switch during this period to conserve energy. From Figure 12, we can see SAML saves 66.2% of energy on the sink node (SAML: 388.6 J and pure TDMA: 1148.2 J) and 30.1% of energy on the sensor node (SAML: 803.6 J and pure TDMA: 1150.0 J) by switching to BoX-MAC on campus (where the interference level is not overly high) and outdoor and switching to RI-MAC at home (high interference due to many WiFi networks). Overall, SAML saves 31.6% of energy (1451.7 J) comparing with pure TDMA but still achieves the QoS reliability requirement of the application throughout the 10-hour case study.

VII. Related Work

Many approaches have been proposed to reconfigure a WSN by disseminating code to the nodes. Hui et al. [14] proposes a reliable dissemination protocol that distributes an entire TinyOS image compiled with a new MAC protocol, across the network and replaces the current running image on the nodes across the network by reprogramming them. By distributing an image nodes benefit from having only the pieces of software that they require at a given time, but the network pays a large communication overhead during the distribution and the nodes incur a large shut down and load times. Many efforts have been made to reduce the communication overhead. Marron et al. [21] proposes an adaptive cross-layer framework that selects and disseminates only new fragments of code instead of an entire image. Mottola et al. [25] uses a reconfiguration programming model to identify a subset of nodes that should be reconfigured, avoiding flooding to the entire network for each image distribution. Tavakoli et al. [34] designs an interval-cover graph to minimize communication redundancies between multiple applications on shared sensors, while Gauger et al. [10] designs an approach to exchange and relink nesC components by defining an uniform external interface for all changeable nesC components. In contrast to these dissemination approaches our research investigates the efficiency and effectiveness of hosting multiple MACs and enabling much more dynamically switching between them at runtime. Our measurements of code size increase of the OS (currently we have results only for TinyOS) resulting from making available numerous MACs shows that it is only marginally larger than an OS image containing a single protocol. Our measurements also suggest that the resulting code size is a non-linear, decreasing function of the number of MACs RMA makes available.

Recently, Zhang et al. [36] designs a toolchain to enable the MAC reconfiguration by relinking MAC components at runtime. This design takes a different approach than our RMA: A computer is used to compute a MAC description, a parser processes the description and generates a new MAC based on preloaded MAC components of the sensors participating in the network. The MAC is then sent to the sensors through radio communication. MAC switching therefore requires more resources than RMA (e.g. a nearby computer, processing time, and radio bandwidth) and is can be mobile only to the extent that the computer is mobile. Furthermore, the user is responsible for initiating the work on the computer side to issue the MAC switching command, while RMA determines which protocol is optimal for a given scenario and performs the switch automatically.

The research community has a growing interest in designing hybrid MACs to combine some of the advantages of different MAC protocols. Z-MAC [29] allows nodes to compete for the channel within unassigned TDMA slots. Funneling-MAC [3] allows nodes close to the sink to run TDMA schedules while all other nodes follow a scheduled contention or polling based duty cycle. However, these hybrid MACs provide limited number of features, require tremendous design efforts, sometimes without full benefits of optimality. For example, both Z-MAC and Funneling-MAC are not as resilient to interference as receiver-initiated MACs. There are also growing interest in adding adaptation to a single, static, protocol at run time. IDEA [5] and MaxMAC [15] proposed to extend battery lifetime and accommodate temporal higher traffic demand by adjusting the wakeup interval of LPL BoX-MAC. pTunes [38] allows for runtime adaptation of MAC parameters. In contrast to the hybrid MACs, which provide limited number of features, and parameter tuning approaches, which optimizes the parameters of a single MAC – our work investigates the possibility of switching between various MACs, while enjoying the full benefit of a specific protocol to given conditions. Reasonably, this offers applications a richer optimization space.

Adaptive MACs were also research in relation to IEEE 802.11 networks. Doerr et al. [7] designs an adaptive MAC Framework which loads multiple MACs and dynamically chooses one. Huang et al. [13] develops an adaptive MAC protocol which can select between multiple MACs. Farago et al. [9] proposed to dynamically combine a set of existing MACs into a single layer. The major difference between these 802.11 dynamic MAC frameworks is that our work is designed to minimize run time overhead and memory footprint. Factors which are critical in low power and resource constrained sensor networks. Our measurements and empirical results suggest that our approach of component reuse, and variable changes to perform MAC switches, effectively reduce static code size and runtime overhead. To the best our knowledge, RMA is the first reconfigurable MAC architecture that allows dynamic MAC switching for low power WSNs.

We have introduced some high-level concept of SAML and a simple proof-of-concept integration of two basic MACs in [17]. Comparing with [17], this paper presents the detailed design of our Self-Adapting MAC Layer comprising Reconfigurable MAC Architecture and MAC Selection Engine as well as a systemic experimental evaluation.

VIII. Conclusion

The integration of smart phones and wireless sensors expose the MAC layer to dynamic environments and varying applications where a fixed MAC protocol cannot always deliver satisfactory performance. We present SAML, a Self-Adapting MAC Layer that enables wireless sensors to switch the MAC at run time. Our experimental results show that SAML provides an efficient and reliable MAC switching, and it adheres to the application specified requirements.

References

[1] http://www.tinyos.net/.

[2] Accessory Development Kit. http://developer.android.com/tools/adk/index.html.

[3] G.-S. Ahn, S. G. Hong, E. Miluzzo, A. T. Campbell, and F. Cuomo. Funneling-mac: a localized, sink-oriented mac for boosting fidelity in sensor networks. In *SenSys*, 2006.

[4] C. A. Boano, M. A. Zuniga, K. Roemer, and T. Voigt. Jag: Reliable and predictable wireless agreement under external radio interference. In *RTSS*, 2012.

[5] G. W. Challen, J. Waterman, and J. Waterman. Idea: Integrated distributed energy awareness for wireless sensor networks. In *MobiSys*, 2010.

[6] O. Chipara, C. Lu, T. Bailey, and G.-C. Roman. Reliable clinical monitoring using wireless sensor networks: Experience in a step-down hospital unit. In *SenSys*, 2010.

[7] C. Doerr, M. Neufeld, J. Fifield, T. Weingart, D. C. Sicker, and D. Grunwald. Multimac-an adaptive mac framework for dynamic radio networking. In *DySPAN*, 2005.

[8] C. E. K. et al. Recommendations for Standardization of Leads and of Specifications for Instruments in Electrocardiography and Vectorcardiography. XXXV:583–602, March 1967.

[9] A. Farago, A. D. Myers, V. R. Syrotiuk, and G. V. Zaruba. Meta-mac protocols: Automatic combination of mac protocols to optimize performance for unknown conditions. In *IEEE Journal on Selected Ares in Comunication*, 2000.

[10] M. Gauger, P. J. Marron, and C. Niedermeier. Tinymodules: Code module exchange in tinyos. In *INSS*, 2009.

[11] Hardware design of TelosB. http://webs.cs.berkeley.edu/tos/hardware/telos/telos-revb-2004-09-27.pdf.

[12] F. Hermans, O. Rensfelt, T. Voigt, E. Ngai, L.-A. Norden, and P. Gunningberg. Sonic: Classifying interference in 802.15.4 sensor networks. In *IPSN*, 2013.

[13] K.-C. Huang, X. Jing, and D. Raychaudhuri. Mac protocol adaptation in cognitive radio networks: An experimental study. In *ICCCN*, 2009.

[14] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *SenSys*, 2004.

[15] P. Hurni and T. Braun. Maxmac: a maximally traffic-adaptive mac protocol for wireless sensor networks. In *EWSN*, 2010.

[16] M. Keally, G. Zhou, G. Xing, J. Wu, and A. Pyles. Pbn: Towards practical acitvity recognition using smartphone-based body sensor networks. In *SenSys*, 2011.

[17] T.-S. Kim, T. Park, M. Sha, and C. Lu. Toward mac protocol service over the air. In *GLOBECOM*, 2012.

[18] K. Klues, G. Hackmann, O. Chipara, and C. Lu. A component-based architecture for power-efficient media access control in wireless sensor networks. In *SenSys*, 2007.

[19] P. Levis and D. Gay. Tinyos programming. April 13, 2009.

[20] K. Lin and P. Levis. Data discovery and dissemination with dip. In *IPSN*, 2008.

[21] P. J. Marron, D. Minder, A. Lachenmann, and K. Rothermel. Tinycubus: An adaptive cross-layer framework for sensor networks. In *Information Technology*, volume 47, pages 87–97, 2005.

[22] MLA Source Code. http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x-contrib/wustl/.

[23] Monsoon Solutions. http://www.msoon.com/LabEquipment/PowerMonitor/.

[24] D. Moss and P. Levis. Box-macs: Exploiting physical and link layer boundaries in low-power networking. Technical Report SING-08-00, Stanford University, 2008.

[25] L. Mottola, G. P. Picco, and A. Amjad. Figaro: Fine-grained software reconfiguration for wireless sensor networks. In *EWSN*, 2008.

[26] Polar Heart Rate Monitor. http://www.polarusa.com/.

[27] J. R. Quinlan. C4.5: Programs for machine learning. In *Morgan Kaufmann Publishers*, 1993.

[28] Z. Ren, G. Zhou, A. Pyles, M. Keally, W. Mao, and H. Wang. Bodyt2: Throughput and time delay performace assurance for heterogeneous bsns. In *INFOCOM*, 2011.

[29] I. Rhee, A. Warrier, M. Aia, and J. Min. Z-mac: a hybrid mac for wireless sensor networks. In *SenSys*, San Diego, California, USA, 2005.

[30] M. Sha, G. Hackmann, and C. Lu. Energy-efficient low power listening for wireless sensor networks in noisy environments. In *IPSN*, 2013.

[31] M. Sha, G. Hackmann, and C. Lu. Real-world empirical studies on multi-channel reliability and spectrum usage for home-area sensor networks. *IEEE Transactions on Network and Service Management*, 10(1):56–69, 2013.

[32] Shimmer Wireless ECG sensor. http://www.shimmer-research.com/.

[33] Y. Sun, O. Gurewitz, and D. B. Johnson. Ri-mac: A receiver-initiated asynchronous duty cycle mac protocol for dynamic traffic loads in wireless sensor networks. In *Sensys*, 2008.

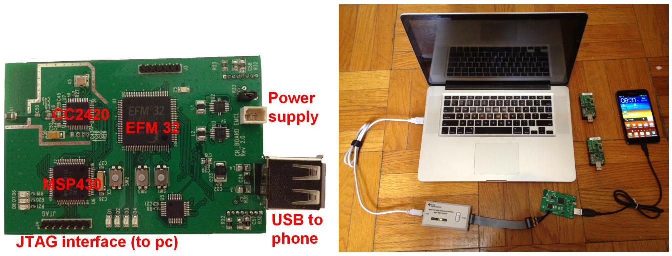
[34] A. Tavakoli, A. Kansal, and S. Nath. On-line sensing task optimization for shared sensors. In *IPSN*, 2010.

[35] Weka. http://www.cs.waikato.ac.nz/ml/weka/.

[36] X. Zhang, J. Ansari, L. M. A. Martinez, N. A. Linio, and P. Mahonen. Enabling rapid prototyping of reconfigurable mac protocols for wireless sensor networks. In *WCNC*, 2013.

[37] ZigBee 2007 specfification, 2007. http://www.zigbee.org/Standards.

[38] M. Zimmerling, F. Ferrari, L. Mottolay, T. Voigty, and L. Thiele. ptunes: Runtime parameter adaptation for low-power mac protocols. In *IPSN*, 2012.



(a) Gateway board. (b) System.

Fig. 13. Gateway board connects with a smart phone through USB and with a laptop through JTAG interface.

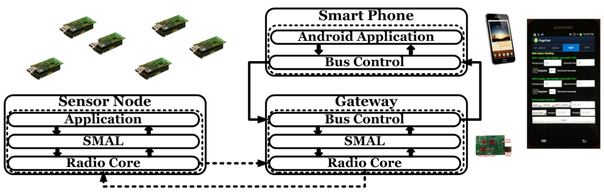


Fig. 14. Software architecture of our platform.

Appendix I: Gateway for Smart Phones

This section introduces our gateway board that enables an Android phone to communicate with a 802.15.4 radio.

To design the gateway board (shown in Figure 13(a)) we follow the open source design of TelosB [11] and add an external clock to increase the baud rate of the serial interface between mote OS and the Android OS. This board is connected to the smart phone's mini USB port and can communicate with sensors wirelessly over its IEEE 802.15.4 radio. An EFM 32 chip is added for future extension and MUXing multiple data flows generated by multiple 802.15.4 networks. In our current implementation the EFM 32 forwards all packets sent and received by the gateway board's MSP430 and the phone CPU chip (e.g. the MSM8660 platform on the Samsung Galaxy note).

To support the communication between the EFM32 chip and the MSP430 microcontroller, we utilize the UART interface which is already available in TinyOS library. Any data between MSP430 and EFM32 is delivered through this UART.

We also included a JTAG interface to enable a connection to a computer for programming and debugging. Figure 13(b) shows how the gateway board connects with a laptop and a smart phone.

Figure 14 shows the software architecture of the new platform. The gateway implements SAML for wireless communication with sensors, and relay any data received from the sensors to the Android phone over the USB. On the phone we develop an Android application, which is automatically booted when the smart phone detects a USB connection. The Android Application provides an GUI for user to manually inject the MAC IDs to the gateway through the USB interface using APIs provided by Accessory Development Toolkit (ADK) [2].

Compared with the platforms proposed in literature [16] [28], our platform doesn't require any modification to the smart phone hardware or the Android kernel. This effectively insulate the developments of TinyOS application and Android application, allowing for separation of concerns and providing a convenient platform for mobile sensing applications that integrates wireless sensors and Android phones.