Toward MAC Protocol Service over the Air

Tae-Suk Kim, Taerim Park
Samsung Advanced Institute of Technology
Samsung Electronics
Gyunggi-do, South Korea
Email: {taesuk.kim, taerim.park}@samsung.com

Mo Sha and Chenyang Lu
Department of Computer Science and Engineering
Washington University in St. Louis
Email: {sham,lu}@cse.wustl.edu

Abstract—**With the rapid permeation of smartphones and wireless sensors in our society, smartphones are poised to become *personal hubs* connecting wireless sensors with users and the Internet. Due to frequent changes to applications and network conditions, wireless networks connecting a personal hub and wireless sensors must meet time-varying QoS requirements at minimal energy cost. This paper presents the architecture of a novel *Protocol Service System (PSS)* towards the vision of protocol service over the air. In contrast to traditional wireless networks where a single MAC protocol is statically selected *a priori*, PSS switches among multiple MAC protocols at *run time* to dynamically optimize power efficiency subject and meet current QoS requirements. To meet the memory constraint on wireless sensors PSS employs a component-based reconfigurable MAC architecture to support multiple MAC protocols at significantly reduced memory footprint through component sharing. The feasibility of PSS has been demonstrated through a proof-of-concept implementation of the PSS architecture and the development of a personal hub prototype running the Android OS.**

I. Introduction

We will live with numerous devices around us in the future. It is forecasted that more than 7 trillion devices will be deployed by 2020 [1]. Another report predicted 50 billion consumer electronics including smartphones and 1 trillion sensors including RFID/USN will be connected by 2020 [2]. Recent years have already witnessed remarkable increase in smartphone dissemination. Due to its generality in purpose, processing power comparable to that of PC, and connectivity with both the Internet and personal area networks, smartphone will serve as a *personal hub* connecting wireless sensors with the user and the world. The trend of integration of smartphones and wireless sensors is evident in emerging applications such as mobile health and entertainments.

In contrast to local area networks, personal networks formed by personal hub has two main features. First, devices are required to control energy consumption tightly. In personal networks, most devices are expected to be battery-powered and require a battery life from months to years. Second, a personal network may need to support a wide range of application-specific QoS requirements such as latency and compliance with real-time constraints (e.g., monitoring and control in industrial environments), or reliable data delivery (e.g., medical applications).

In this paper, we propose to address these challenges by changing the MAC protocol at run time in response to changes in application QoS requirements and network conditions. MAC reconfiguration is an effective control knob due to the significant impacts of MAC on network performance and energy consumption. However, dynamics inherent in personal networks poses challenges. As applications, location, user preference and network conditions change over time, a wireless network protocol optimized to a specific scenario can lead to poor power efficiency and QoS. Therefore a personal network connecting the personal hub and wireless sensors needs to be reconfigured at run time in response changing QoS requirements and conditions.

This paper proposes a *Protocol Service System (PSS)* that realizes *protocol service over the air* to support adaptive wireless communication between a personal hub and surrounding sensors. PSS determines the optimal MAC protocol with respect to the current QoS requirements, workload and network conditions and then switches to the selected MAC protocol at run time. Specifically, the main contributions of this paper are three-fold:

First, we propose a novel Protocol Service Architecture that optimizes energy efficiency while meeting the current QoS performance requirements through MAC layer reconfiguration. Our architecture comprises a *Protocol Service Engine (PSE)* for identifying the best MAC protocol and a *Reconfigurable MAC Architecture (RMA)* for dynamically switching MAC protocols. In contrast to traditional wireless networks where a single MAC protocol is statically selected *a priori*, our proposed system dynamically switch among multiple MAC protocols at *run time*.

Second, we propose a component-based approach to Reconfigurable MAC Architecture. A key challenge in supporting multiple MAC protocols on wireless sensors is their limited memory. The proposed architecture minimizes the memory footprint by reusing common components shared by multiple MAC protocols. A proof-of-concept implementation of RMA with two MAC protocols, BoX-MAC and Pure TDMA, achieves 60% and 70 % savings in ROM and RAM, respectively, compared with the sum of the footprints of the two individual protocols.

Third, we present a proof-of-concept implementation of PSS on an Android-based platform. Our personal hub prototype integrates an Android-based mobile device emulating a smartphone and a TinyOS-based mote for communication with wireless sensors. *To our knowledge, this is the first implementation that realizes the idea of protocol service on* *a mobile device.*

II. Background and Related Works

There are some existing approaches for MAC protocol optimization and adaptation. Polaste et al. [3] present an analytical model of node lifetime for B-MAC. Buettner et al. [4] derive models of energy consumption and per-hop latency for X-MAC. Ye et al. [5] present a model of energy consumption to find optimal protocol parameters for SCP. Those approaches are, however, limited to configuration at compile time, which leads to suboptimal performance if the network conditions deviate from the expectations.

Recently, there are some efforts to support MAC optimization in run time. Zimmerling [6] presents a scheme for automatic parameter adaptation for X-MAC. Meier et al. [7] propose a MAC optimization architecture that automatically configures the MAC parameters at runtime in order to increase the network lifetime. However, these works do not support protocol switching (i.e., protocol optimization rather than parameter optimization) in run time.

Among previous works the MAC protocol engine [8] is similar to ours in perspective of protocol optimization. The protocol engine then chooses the optimal protocol among its library, in which the analysis of each protocol is provided, with optimal protocol parameters satisfying constraints for energy, delay and reliability for a given network topology. What is different from our work is that selection of the optimal protocol is performed in off-line and corresponding parameters are set at compile time. There is another work [9] on the concept of protocol optimization. They present a method to dynamically combine any set of existing MAC protocols into a single higher layer, called meta-MAC protocol. However, their concept is not realized and validated through system-level implementation.

III. Protocol Service System

The protocol service system (PSS) actively reconfigures the protocol of the application program. To this end, the system needs to perform the function of *determining* an optimal protocol for connections with neighbor sensor nodes considering applications operated on each nodes and *reconfiguring* the operating protocol to the optimal one, and the function of *receiving* a request from another protocol service system to reconfigure the current protocol. The protocol service system including both functions is referred to as a full functional protocol system device (FPD), while a device only with the function of receiving is referred to as a reduced functional protocol system device (RPD). A FPD consists of an analyzer, a protocol engine, a protocol realizer, and a reconfigurable protocol stack. A FPD is able to play a role of RPD by deactivating modules related with the function of reconfiguring: the analyzer and the protocol engine. Fig.1 illustrates an operation between protocol reconfiguration capable devices, a FPD and a RPD. In the following, we provide the description of its constituent elements and the sequence of operation between elements.

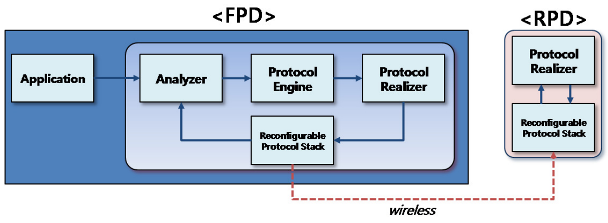


Fig. 1. Protocol reconfiguration system: FPD-RPD communication

A. Analyzer

The analyzer analyzes requirement information of the application program and system information that is obtained from a protocol layer. The requirement information of the application program includes QoS related information of the application program, for example, a latency, a Packet Error Rate (PER), a lifetime, etc. The system information includes information that the PSS may obtain from the underlying protocol layers with respect to a physical (PHY) layer, a media access control (MAC) layer, and the network layer, for example, channel state information, the MAC operational parameter, the number of devices connected. After selecting the necessary information, the analyzer periodically provides the information to the protocol engine in a predetermined message format. During this process, the analyzer can trigger the protocol engine using an event-based method that is executed every time information is received or a period based method of triggering the engine at predetermined intervals.

B. Protocol Service Engine

The protocol engine verifies in real time an optimal protocol of an application program and determines operating parameters of the corresponding protocol. The protocol engine includes a modeling unit and an algorithm unit. The modeling unit formulates an optimization problem with QoS constraints from applications. For each protocol that the system retains, the performance and QoS metrics, such as energy consumption, latency, throughput, etc., are modelled mathematically as a function of parameters that are obtainable from the analyzer. Utilizing those models, with the constraints on the applications' requirements expressed by the models, an optimization problem with a metric of interest can be formulated, for example, energy minimization using the latency requirements. Those modeling is performed in off-line fashion.

The protocol engine obtains, from the analyzer, a value with respect to requirement information of the application program and system information in real-time and then input the obtained value in the problem defined by the modeling unit. The protocol engine then determines operating parameters that optimize the target metric of the underlying protocol within the range in which requirement information of the application program is satisfied. These solving process is performed by the algorithm unit. The algorithm unit selects or develops a suitable algorithm based on characteristics of the optimization problem formulated by the modeling unit. More detail description is provided in section IV.

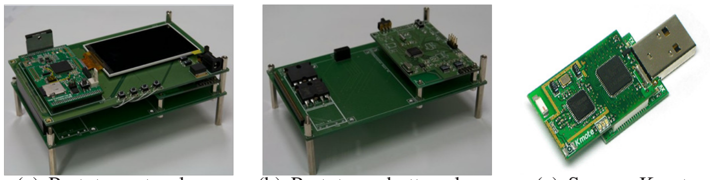

(a) Prototype: top layer (b) Prototype: bottom layer (c) Sensor: Kmote

Fig. 2. HW architecture

C. Realizer

The protocol realizer parses protocol configuration information that is received from the protocol engine, and enables a protocol reconfigured based on the protocol configuration information to be realized in a protocol layer. To this end, the protocol realizer generates and passes information to the protocol stack of local (itself) and remote sensor nodes based on protocol configuration information. Generated information includes a protocol identifier (ID) to be changed to and operating parameter values of the protocol. For the RPD, the protocol realizer receives a protocol configuration request from the underlying protocol stack, which receives the information from the FPD via air. As in the FPD, the information, parsed from the received information, necessary for the underlying stack to reconfigure to the requested protocol, is passed to the stack.

D. Reconfigurable Protocol Stack

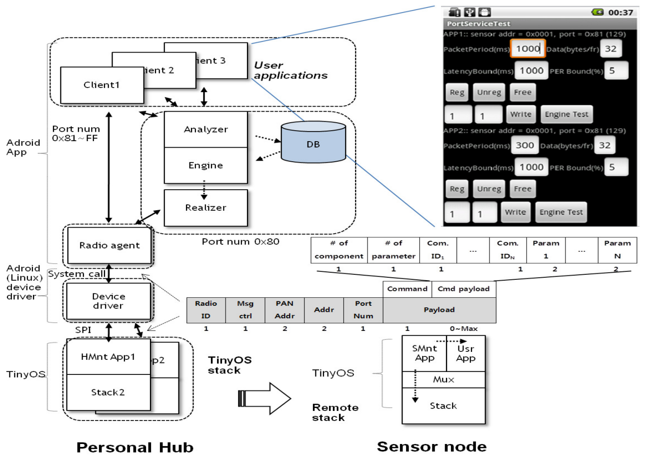
The reconfigurable protocol stack maintains a memory that stores a stack of the plurality of component modules. A specific protocol is composed of corresponding set of component modules; through wiring necessary components a necessary function for the protocol can be executed. The component modules have trade-off due to overhead resulting from granularity and thus, can be variously configured based on a designer's intent. The reconfigurable protocol stack transmits information, received from the realizer, to realize the determined protocol for the application program in the target device, for example, the RFD connected to the protocol reconfiguration system. Realization of the reconfigurable protocol stack is further discussed in section IV.

IV. Implementation

This section describes the implementation of a PSS prototype as a proof of concept. We first present the hardware platform, followed by the software architecture.

A. HW architecture

The system consists of two HW platforms. The first one emulates a mobile device. For this purpose, we developed a HW prototype as shown in Fig.2. For emulating a sensor node we used Kmote (similar to TelosB). The mobile device has two layers. The top layer supports computation functions of a mobile device. In the layer, S5PC110 a low power application processor targeting high performance mobile devices and a 3.7 inch touch screen are equipped. S5PC110 is a 32 bit RISC ARM Cortex-A8 based microprocessor. Its operating frequency goes up to 1GHz and a lot of peripherals for system,



APP1:: sensor addr = 0x0001, port = 0x81 (129)
PacketPeriod(ms) 1000 Data(bytes/fr) 32
LatencyBound(ms) 1000 PER Bound(%) 5

APP2:: sensor addr = 0x0001, port = 0x81 (129)
PacketPeriod(ms) 300 Data(bytes/fr) 32
LatencyBound(ms) 1000 PER Bound(%) 5

# of component	# of parameter	Com. ID_1	...	Com. ID_N	Param 1	...	Param N
1	1	1		1	2		2

Radio ID	Msg ctrl	PAN Addr	Addr	Port Num	Command	Cmd payload
1	1	2	2	1	1	0~Max

Fig. 3. SW architecture

memory, multimedia and connectivity are included. On the microcontroller, we ported Android 2.2. The bottom layer is a system to support communication, mainly short range communication. It adopts WiFi and Bluetooth modules, and they are directly connected to the application processor with SPI and UART through inter-layer connector. For our system, we prepared a separated place for independent modules which may consist of a communication processor and a radio transceiver. Then, we plugged in a Kmote, adopting MSP430 as a communication controller and cc2420 as a transceiver. Also we ported TinyOS 2.1.1 [14] as operating system for reconfigurable protocol stack. It corresponds to communication firmware in view of a mobile operating system. The Kmote is connected to the application in the first layer through a SPI interface.

B. SW architecture

Fig.3 shows SW architecture of the protocol reconfiguration system. The system mainly consists of four software blocks: client, server, radio agent, and reconfigurable stack. Client, server, and radio agent are implemented in Android and reconfigurable stack is implemented on top of TinyOS. In the following, the details on implementation of each software blocks are discussed.

1) Client: Client is an Android activity. Like any other user application, it may get input from a user and present results. For the simple testing, we implemented one client for controlling two separated sensor nodes assuming two independent applications. The client has two text edit boxes (packet per seconds and latency bound), two buttons (register and deregister), and one text view (received contents) for each application as shown in Fig.3. Register and deregister buttons are used to emulate conditions of starting/ending a new service. When the register button is pushed, a message containing packet rates and latency bound is passed to an Android service, the server.

2) Server: In this section, we describe how two main elements of protocol service engine, protocol optimization problem and solver, are implemented.

- **Analyzer:** An analyzer collects information from clients and environments, then parse the collected information for the engine. The information is passed by calling the engine written in C-language through JNI interface. In our current implementation, the analyzer only passes the aggregated packet rates and the smallest latency bound among two applications.
- **Protocol Service Engine - protocol optimization problem:** Based on the analysis of performance metrics for two sensor protocols, BoX-MAC [11] and Pure TDMA (PTDMA) [12] (represented in Appendix), the optimization problem that chooses the best protocol and its operating parameters (in terms of energy consumption), given QoS (latency in this implementation) can be formulated as

$$\min_{p \in \{\texttt{BoX-MAC}, \texttt{PTDMA}\}} \quad E_p^*, \tag{1}$$

where E_p^* is the optimal value for a problem for each protocol that minimizes the energy consumption over feasible operating parameter, which can be formulated as

$$\begin{aligned} \min_{x} \quad & E_p(x) \\ \text{subject to} \quad & L_p(x) \leq Q_l, \end{aligned} \tag{2}$$

where $E_p(x)$ represent the energy consumption of protocol p with the use of parameter x (the wake-up interval (T_w) and the frame length (T_f) for BoXmac and PTDMA, respectively. See Appendix), $L_p(x)$ is the latency incurred by p using the value of x, and Q_l is the QoS requirement for latency. Thus, by solving (2) for each protocol, the optimal parameter achieving the minimum energy consumption for the latency bound are determined. The optimal values are compared and the best protocol is selected through (1). In the next section, we will deal with how to solve the problem.
- **Protocol Service Engine - solver:** the optimization problems above has two properties: first, the problem (2) belongs to integer problem domain (since operating parameters of each protocol is integer), and second, the problem (2) without integer constraint is the convex problem since the objective and requirements are power function. Exploiting those properties, we utilize *branch & bound* algorithm [10] to solve the problem (2), which is a general approach to integer problem.
- **Realizer:** Realizer configures both local and remote TinyOS protocol stacks through issuing control message. The procedure for configuring protocol stacks managed by the realizer is as follows: (1) Stop applications of remote sensors, (2) Request protocol configuration and confirmation with all neighboring sensor nodes, (3) Configure local protocol stack, and (4) Restart remote applications. Control message format with the procedure above will be discussed in the section of radio agent below.

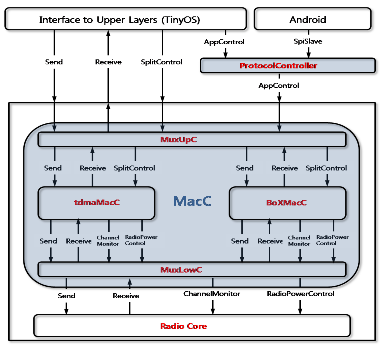


Fig. 4. Reconfigurable MAC Architecture

3) Reconfigurable MAC Architecture: The Reconfigurable MAC Architecture (RMA) prototype is implemented in TinyOS and runs on the MSP430 microcontroller which is connected to the S5PC110 and a Chipcon CC2420 (compliant with IEEE 802.15.4) via two SPIs. RMA is implemented based on the component-based MAC Layer Architecture (MLA) [12]. While the current implementation of RMA only supports two MAC protocols, BoX-MAC and PTDMA, it can be extended to support more MAC protocols using the MAC components in MLA. By keeping only a single copy of each common component shared by multiple MAC protocols, RMA can support a multitude of MAC protocols without incurring excessive memory cost.

MLA is a component-based architecture for power-efficient MAC protocols. MLA distills various features common to existing MAC protocols into a set of reusable components, optimized for the specific function they are intended to provide. Some of these components are low level, encapsulating the intricacies of a particular hardware platform. Others are high level, providing various functionality typical of MAC protocol design in a reusable fashion. MLA defines link-layer interfaces that allow for a separation between core radio functionality and the logic required to perform radio power management. While MLA is designed to facilitate development of MAC protocols, its MAC components provide the building blocks for RMA to support *run-time* reconfiguration of the MLA layer in a memory-efficient fashion.

We now describe the current implementation of RMA for Box-MAC and PTDMA. Fig.4 shows the MAC container (`MacC`) of RMA. `MacC` encapsulates components of different MAC protocols: `tdmaMacC` and `BoXMacC` encasulate the hardware-independent components for PTDMA and BoX-MAC, respectively. Notably, both MAC protocols share a *single* copy of the radio core resulting in significant savings in memory footprint (as shown in our evaluation). For the MAC protocols to be interfaced with upper/lower layer, two MUX layers are developed in the container; `MuxUpC` component implements the MUX layer between upper layer and MAC layer and `MuxLowC` component implements the MUX layer between MAC layer and the radio core. In the MUX layers, the interfaces from upper layer/radio core are connected to all the MAC protocols in `MacC`. The `ProtocolController` is a component to manage protocol reconfiguration. `ProtocolController` is interfaced with Android via `SpiSlave` interface, with which control information (the protocol request information) is received from protocol service engine, and application data from/to remote sensor node is relayed to/from Android side. `ProtocolController` is also interfaced with `MacC` through `AppControl` interface. Once the protocol request information (ID of the protocol and the values of operating parameters) is received, it is relayed to the MUX layer through `AppControl`. The MUX layer then directs the events/commands to the interface of the selected protocol to communicate with upper/lower layers.

4) Radio agent: Interfaced with Android device driver that is located at the boundary between Android and TinyOS, radio agent is charge of multiplexing and demultiplexing packets among a device driver, clients and the server. For the interface between radio agent and the SW blocks, the MAC payload structure of IEEE 802.15.4 [13] is utilized. We define **Msg ctrl** field (see Fig.3), and according to value of the filed packets are separated into data and control packet for local protocol and parameter configuration and for remote protocol and parameter configuration. In the case of configuring remote stacks, the protocol server needs to control the switching operation of remote sensors. For this purpose, several commands are reserved in the **Command** filed in the payload: application start, application stop, and protocol configuration request.

V. EVALUATION

In this section we show the possible benefit of proposed protocol service system over realistic scenarios. In particular, We evaluate the effectiveness of the protocol reconfiguration system from the perspective of energy gain through (1) parameter optimization on a single MAC protocol and (2) protocol optimization over multiple protocols (BoX-MAC and PTDMA).

A. Experimental set-up

For realistic performance evaluation, we categorize applications in terms of the representative characteristics of applications (data sampling rate and delay requirement) into three application groups. Application group 1 (AG 1): high sampling rate & delay sensitive application, application group 2 (AG 2): low sampling rate & delay sensitive application, and application group 3 (AG 3): low sampling rate & delay insensitive application. Table I shows the values of parameters used in the experiments for each application group. During evaluation of the parameter optimization performance, the prototype is connected with one Kmote sensor node, and the sensor node sends packets, generated by sampling rate of each application instance, to the prototype. Experiments duration for each instance has a length of 100 sampled packets.

For the protocol optimization evaluation, a scenario that simulates a real life usage pattern are considered by dynamically switching application usage; Room temperature sensor (application group 3) is communicated with smartphone during 8 hours sleep. After wake-up, the user goes to the school, attaching ECG sensor on body and carrying a note book with key fob (application group 2). After 2 hours, he comes back to the home and uses ECG only (application group 1). Sensors are coordinated to send packet at the rate of application instance specified in Table I according to the schedule of the scenario. For this evaluation, the two Kmote sensor nodes are connected to the prototype on the fly; when only one application instance is serviced, then only one of the sensor node is serviced. In all the experiments, the energy consumption on each device is measured by power monitor [15].

TABLE I
APPLICATION GROUP INSTANCES

	AG 1	AG 2	AG 3
Sampling rate (packets/sec)	3	1/3	1/30
Latency requirement (ms)	500	200	1000
Example applications	Health-care applications	Security applications	Environmental applications

B. Results

1) Effect of parameter optimization: For the performance comparison with the BoX-MAC with wake-up interval directed by the proposed system, we choose the BoX-MAC using the fixed wake-up interval over all the three application instances. In order to focus on energy consumption gain, we isolate the feasibility for latency requirement by setting the fixed wake-up interval to the level satisfying the most tight delay bound (100 ms for the instance of application group 2), and the wake-up interval is then set to 72 ms using equation (3), the worst case of packet delay. The evaluation results are summarized in Table II, which shows the consumed energy in mAh (the first value in parenthesis) and wake-up interval in ms (the second value) for various protocols. For the instance of application group 1, the wake-up interval, calculated by the protocol engine, is set to 104 ms, which is 30 ms longer than the fixed interval of 72ms. This makes the energy consumption of the sensor node (transmitter) less than that of BoX-MAC with the fixed interval; this is encouraging results in that the impact of the energy consumption on the sensor node is much bigger than the smartphone because the battery size of the sensor node is much smaller than that of the smartphone. For the case of application group 2, the feasible wake-up region to the delay bound (100 ms) is $[0, 72]$, and parameters of both protocols are set to the maximum feasible point, achieving the maximum energy saving of receiver. We can easily expect the better energy consumption would be achieved with a point out of the feasible region but that would degrade the QoS due to unsatisfied delay. The performance difference becomes stark in the case of application group 3. The protocol reconfiguration system results in 972 ms of wake-up interval, achieving the 598% energy saving compared to the BoX-MAC with 72 ms interval.

2) Effect of protocol optimization:

- **Switching Overhead:** The overhead occurred in the process of protocol reconfiguration are (1) TX/RX of the three control messages (application stop request, protocol

TABLE II
EFFECT OF PARAMETER OPTIMIZATION

	AG 1	AG 2	AG 3
RX (proposed)	(30.82, 104)	(321.52, 72)	(328.42, 972)
RX (BoX-MAC)	(40.21, 72)	(321.41, 72)	(3235.79, 72)
TX (Proposed)	(61.65, 104)	(359.35, 72)	(573.92, 972)
TX (BoX-MAC)	(58.79, 72)	(359.71, 72)	(3059.90, 72)
Sum (Proposed)	(92.47, 104)	(681.02, 72)	(902.34, 972)
Sum (BoX-MAC)	(99.00, 72)	(681.12, 72)	(6295.69, 72)

reconfiguration request, and application start request) and (2) operation invoked on protocol stack for switching protocol. We investigate the switching overhead by measuring the energy consumption caused by them for the switching from BoX-MAC to PTDMA, and PTDMA to BoX-MAC case. In the case of switch from BoX-MAC to PTDMA, a sensor node spends the extra 0.13 μAh receiving a control message. The energy consumption of processing on switching operation is observed to be 4.53 μAh consuming 935.12 ms, during which, BoX-MAC related functions are turned down, PTDMA functions are turned on, and synchronizing to coordinator's beacon is performed. For the case of switching from PTDMA to BoX-MAC, the control message invokes 0.01 μAh extra energy consumption, and energy consumption of 2.31 μAh for switching processing (449 ms). Fig.5 represents the energy consumption behavior of receiving a control message and switching processing measured at the sensor node for the two switching cases considered. Considering the order of energy consumption of daily usage amounts to scores to hundreds of mA as shown in the results below, the amount of overhead is very slight.

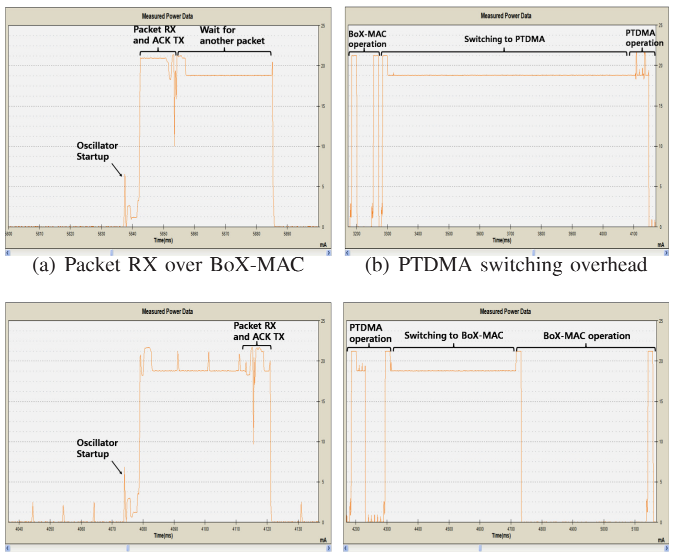
- **Performance:** The consumed energy of various protocols executed for the scenario is summarized in Table III. As the experiment progresses, the protocol service system selects the BoX-MAC with the wake-up interval of 972 ms for temperature application, the BoX-MAC with the wake-up interval of 72 ms for ECG and key fob, and the PTDMA with the frame length of 490 ms for ECG application. With this selection the protocol service system achieves 157% and 322% longer battery life against BoX-MAC and PTDMA with fixed parameters.

TABLE III
EFFECT OF PROTOCOL OPTIMIZATION

	node	AP 3 (8 h)	AP 1 + AP 2 (2 h)	AP 1 (3 h)
BoX-MAC	node 1	(30.85, 72)	(18.25, 72)	(7.14, 72)
	node 2		(10.36, 72)	
PTDMA	node 1	(70.87, 90)	(17.77, 90)	(7.02, 90)
	node 2		(17.24, 90)	
Proposed	node 1	(5.92, 972 (B))	(9.13, 72 (B))	(1.90, 490 (P))
	node 2		(5.18, 72 (B))	

VI. CONCLUSION

We envision that mobile devices such as smartphones will interact with wireless sensors "closely and intelligently" as do current smartphones with human. Personal networks connecting mobile devices and sensors must meet application-specific QoS requirements at minimal energy in dynamic environments. This paper proposes a *Protocol Service System (PSS)* that maintains power efficiency and desired time-varying QoS by dynamically switching MAC protocols at run time. We have demonstrated the feasibility and efficacy of PSS through a proof-of-concept implementation of the PSS architecture and the development of an Android-based mobile platform supporting PSS. In the future we will generalize our PSS to support more MAC protocols and *backward compatibility* with legacy wireless sensors. This work represents a promising step towards the convergence of smartphones and wireless sensors through universal and optimized connectivity.



(a) Packet RX over BoX-MAC (b) PTDMA switching overhead

(c) Packet RX over PTDMA (d) BoX-MAC switching overhead

Fig. 5. Switching Overhead

APPENDIX

In this section, protocol behavior of BoX-MAC and PTDMA is first introduced and analysis of energy consumption and latency for each protocol is driven.

A. BoX-MAC

BoX-MAC is an asynchronous MAC protocol; A receiver node periodically wakes up at intervals of T_w to check incoming packets. Once a data has arrived[1], a transmitter starts initial back-off with maximum back-off window size, T_{bw}. At the expire of the back-off window, if the medium comes to idle during the carrier sense interval of T_{cs}, it transmits a data packet as a preamble to wake up the receiver, and it takes time of T_{pkt}. After sending the packet, the transmitter waits for the ACK packet from the receiver during the duration of T_{al}. If the transmitter does not receive the ACK, it performs carrier sense after short back-off, T_{sbw}, and retransmits the packet if the channel is free. If energy is detected as a result of carrier sense when the receiver wakes up, the receiver keeps radio on and wait for the next *entire* packet. Once the packet is successfully received, the receiver transmits the ACK packet with the air time of T_{ack}.

[1]we assume that packet arrival process is *i.i.d.*

1) Energy-consumption: Energy efficiency is analyzed by considering the different sources individually. Energy is spent performing periodic carrier senses (E_{cs}), sending (E_{tx}), and receiving (E_{rx}). When receiving, the node receives $T_{pkt}/2$ of the packet in average before the first packet is received successfully. For the case of transmitting, the sender has $T_w/(T_{pkt}+T_{al})$ transmission chances, and half of them times the packet is transmitted assuming the packet arrival is *i.i.d.* Energy consumption during T_w is

$$\begin{aligned}
E_{cs} &= P_{rx}T_{cs}/T_w \\
E_{rx} &= \lambda((T_{al}+T_{sbw}/2+3T_{pkt}/2)P_{rx}+T_{ack}P_{tx}) \\
E_{tx} &= \lambda((T_{bw}/2+T_{cs})P_{rx}+(T_w/(T_{pkt}+T_{al})/2) \\
&\quad \cdot T_{pkt}P_{tx}+(T_w/(T_{pkt}+T_{al})/2)T_{al}P_{rx}) \\
E_{\texttt{BoX-MAC}} &= E_cs+E_rx+E_tx
\end{aligned}$$

2) Latency: Latency is defined as the elapsed time from when the packet arrives at the protocol layer to when a destination receives the packet. Considering more packets within latency requirement, higher user satisfaction, we perform the worst case analysis for the latency that a packet experiences. We assume as soon as a packet has arrived at protocol layer, the packet is allowed to be sent right away, i.e., no queuing delay due to prior packet. The worst case scenario is the case where a transmitter waits for the max back-off windows and transmits the packet right after the receiver goes to sleep. Latency, $L_{\texttt{BoX-MAC}}$, in this case is expressed as

$$L_{\texttt{BoX-MAC}} = T_{bo}+T_{cs}+T_w+T_{pkt}. \tag{3}$$

B. Pure TDMA

PTDMA is a TDMA protocol designed for a single-hop network in which nodes communicate to a single base-station. This protocol is similar to the GTS portion of 802.15.4. A frame contains both active and sleep periods. The length of each period can be configured by the application. The first slot in the active period is reserved for exchanging time synchronization information. The remainder of the slots in the active period is assigned to nodes for transmitting packets without contention. All nodes will be on during its own active slots since they may be receiving packets. Radio is waked up in the first slot of the active period and turned off during sleep period.

1) Energy-consumption: Energy is spent performing periodic sending a beacon from a hub ($E_{bc}^{\langle hub\rangle}$), receiving a beacon ($E_{bc}^{\langle node\rangle}$), sending and receiving of a hub in active period ($E_{active}^{\langle hub\rangle}$), and sending and receiving of a node in active period ($E_{active}^{\langle hub\rangle}$). A hub sends a beacon packet in T_{bc} during the first slot, T_{slot}, in every frame, T_{frame}. In the active period, T_{active}, a node transmit a packet in an assigned slot, and during the remainder of the active interval it is listening for packets to itself. Energy consumed during T_f is

$$\begin{aligned}
E_{bc}^{\langle hub\rangle} &= (P_{tx}T_{bc}+P_{tx}(T_{slot}-T_{bc}))/T_f \\
E_{bc}^{\langle node\rangle} &= P_{rx}T_{slot}/T_f \\
E_{active}^{\langle hub\rangle} &= P_{rx}(T_{active}-T_{slot})/T_f \\
E_{active}^{\langle node\rangle} &= (P_{rx}(T_{active}-T_{slot}-T_{pkt})+P_{tx}T_{pkt})/T_f \\
E_{\texttt{PTDMA}} &= E_{bc}^{\langle hub\rangle}+E_{bc}^{\langle node\rangle}+E_{active}^{\langle hub\rangle}+E_{active}^{\langle node\rangle}
\end{aligned}$$

2) Latency: The worst case is when the packet has just arrived right after the assigned slot for the transmitter starts. In this case, the packet stays in queue during T_{frame}, waiting for the slot assigned in the next frame. The latency can be then expressed as

$$L_{\texttt{PTDMA}} = T_f+T_{pkt}. \tag{4}$$

References

[1] L. David, D. Dixit, N. Jefferies, "2020 Vision," *IEEE Vehicular Technology Magazine*, vol.5 , no. 3, pp. 22-29, 2010.

[2] "More than 50 billion connected devices," Ericsson white paper [online]. Available: http://www.ericsson.com/res/docs/whitepapers/wp-50-billions.pdf.

[3] J. Polastre, J. Hill, D. Culler, "Versatile low power media access for wireless sensor networks," *In Proceedings of 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys)*, 2004.

[4] M. Buettner, G. Anderson, E. Han, "X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks," *In Proceedings of 4nd ACM Conf. Embedded Networked Sensor Systems (SenSys)*, 2006.

[5] W. Ye, F. Silva, and J. Heidemann,"Ultra-low duty cycle mac with scheduled channel polling," *In Proceedings of 4nd ACM Conf. Embedded Networked Sensor Systems (SenSys)*, 2006.

[6] M. Zimmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele, "pTunes: Runtime Parameter Adaptation for Low-power MAC Protocols," *In Proceedings of the 11th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2012.

[7] A. Meier, M. Woehrle, M. Zimmerling, and L. Thiele, "ZeroCal: Automatic MAC Protocol Calibration," *In Proceedings of the 6th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2010.

[8] S. Ergen, P. Marco, C. Fischione, "MAC Protocol Engine for Sensor Networks," *In Proceedings of IEEE Global Telecommunications Conference (GLOBECOM)*, 2009.

[9] A. Farago, A. Myers, V. Syrotiuk, and G. Zaruba,"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. 16701681, September 2000.

[10] S. Boyd, L. Vandenberghe, "Convex Optimization," Cambridge University Press, 2004.

[11] D. Moss and P. Levis,"BoX-MACs: Exploiting Physical and Link Layer Boundaries in Low-Power Networking," Technical Report SING-08-00, Stanford University.

[12] K. Klues, G. Hackmann, O. Chipara and C. Lu, "A Component-Based Architecture for Power-Efficient Media Access Control in Wireless Sensor Networks," *In Proceedings of ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2007.

[13] IEEE std 802.15.4-2006: Wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (WPANs).

[14] [online] Available: http://docs.tinyos.net

[15] [online] Available: http://www.msoon.com/LabEquipment/PowerMonitor/