Enabling Rapid Prototyping of Reconfigurable MAC Protocols for Wireless Sensor Networks

Xi Zhang, Junaid Ansari, Luis Miguel Amoros Martinez, Noemi Arbos Linio and Petri Mähönen
Institute for Networked Systems, RWTH Aachen University, Kackertstrasse 9, D-52072, Aachen, Germany
Email: xzh@inets.rwth-aachen.de

***Abstract*—Various emerging applications enabled by wireless sensor networks demand new and versatile MAC algorithms. These MAC protocols should be able to adapt to varying application requirements and deployment conditions. In this paper, we present a component based scheme for enabling fast prototyping of MAC protocols using a lightweight toolchain. Our toolchain allows run-time MAC protocol reconfiguration by controlling the data path and the execution flow of the constituent components. Our approach of composing protocols based on common MAC components aims at reducing the protocol prototyping efforts. Moreover, our toolchain implemented for various sensor nodes provides a platform independent MAC meta-language which simplifies the protocol designing, testing and debugging processes. Empirical tests conducted on commercially available sensor nodes show that our approach enables run-time reconfiguration for MAC protocols with affordable memory consumption and execution overhead.**

I. Introduction

Wireless sensor networks are being deployed in a wide variety of applications, especially in embedded networking domain. Data reliability, latency tolerance, volume of generated traffic, and expected life-times vary hugely from one application to another. Hence, there is a need for different types of sensor network MAC protocols supporting different applications and deployment conditions. Developing new MAC protocols is, however, a cumbersome task and typically requires domain expertise and platform specific programming skills. Moreover, the time and effort required for testing and debugging newly developed code on embedded sensor nodes are high. Most of the existing protocols are developed with a monolithic programming style, which restricts code reusability and portability for multiple platforms.

In our ongoing research towards realizing a flexible MAC framework and enabling fast prototyping [1]–[3], we have followed a component based design approach and decomposed MAC protocols into their fundamental functionalities. These reusable components are exposed through well-defined generic APIs, and can be used as building blocks for composing MAC schemes. We have also developed a user friendly graphical Integrated Development Environment (IDE) where a developer can design protocols in the form of flowcharts. Corresponding to the re-loadable and modifiable flowcharts, the MAC code for the target platforms is automatically generated and uploaded to sensor nodes [3]. This approach enables fast prototyping and reduces debugging efforts. Component based approach for enabling reconfiguration has been investigated in areas such as operating systems [4] and network architectures [5]. In this work, our focus is on achieving run-time reconfigurable MAC protocols.

Static and rigidly deployed MAC schemes are unable to adapt to the changing application demands and networking conditions [6]. In our prior work, we have introduced a set of tools for our component based MAC design with run-time reconfiguration capability [2]. The toolchain executes the state machine of a MAC scheme by linking the constituent MAC components at run-time and controlling the data path and the execution flow. As part of the toolchain, we have developed a MAC meta-compiler which is able to compile and correspondingly map the MAC meta-code at run-time on the target platform.. Our developer-friendly MAC meta-language syntax allows expressing even complex MAC protocols in a few lines of code. The MAC meta-language is able to express different types of dependencies and MAC specific intricacies, which eases the development [7].

In this paper, we present a set of tools designed to address the resource constraints of sensor networks while allowing fast prototyping and run-time reconfiguration. We compare our implementation to the hand coded monolithic implementation for different well-known MAC protocols on commercially available sensor nodes. We show that while enabling flexibility and reconfigurability, the execution time overhead is less than 1.4% for our measured worst case. The time required for protocol reconfiguration has been reduced significantly by up to 60% compared to the conventional reprogramming approach. The ability to redirect, add, delete and modify components in the state machine on-the-fly allows run-time protocol reconfiguration of a sensor network to be easily realized by over-the-air programming.

The rest of the paper is organized as follows: In Section II we review the related research work. Section III presents the system design and the implementation details. Section IV presents the empirical performance evaluation on COTS sensor nodes for well-known MAC protocols. Finally we conclude in Section V.

II. Related Work

The learning curve for code development on sensor nodes can be steep because of the need for new languages, tools and programming paradigms. There has been attempts to simplify the programming effort by providing Java compatible virtual machines such as Darjeeling [8] and Mote Runner [9]. Java applications are converted to bytecode which can be interpreted

978-1-4673-5939-9/13/$31.00 ©2013 IEEE

on sensor nodes. However, the interpretation cost of Java as compared to native code is quite expensive [8]. Ellul *et al.* [10] have lowered the computational requirements on sensor node by translating an intermediate bytecode to native code which reduces the interpretation overhead. Although these schemes lower the programming burden, the run-time reconfiguration issue is not addressed. In addition, the Java based tools target only at programming sensor network applications. They might not fit to the real-time programming requirements and strict timelines imposed by MAC protocol development and reconfiguration.

In the meanwhile, efforts from a different angle have been made to ease the implementation burden and maximize code reuse by defining a unified protocol structure. A unified link layer abstraction has been proposed to implement a broad range of networking and data link technologies without significant loss of efficiency [11]. A Unified Link-Layer API (ULLA) [12] offers a common interface to retrieve link layer information independent of the underlying hardware. Unified Power Management Architecture (UPMA) [13] and its extension [14] separates power management features from the core radio functionality to plug it easily into different hardware specific MAC implementations. The component based MAC Layer Architecture (MLA) [15], [16] further extends the UPMA idea by decomposing platform-independent part of MAC protocols into reusable components. Significant code reuse across different protocols with a low memory overhead and without significant loss in terms of performance metrics has been achieved. However, the hardware and protocol specific code contributes heavily to the MAC implementation effort. In MAC-PD [3] we identified the basic functional components (cf. Section III-A) which can be used not only to form existing protocol genre but also non-classical designs. In addition, MAC-PD eases MAC design through a drag & drop based graphical user interface [17]. However, this tool does not support run-time reconfiguration without reprogramming nodes. Additionally, although the philosophy of designing MAC protocols in the form of flowchart can be favourable for certain groups of designers, some programmers prefer "code-like" syntax.

There has been recent research activities in realization flexible MAC schemes in the area of less resource constrained networks. Bianchi *et al.* discuss the advantages of adaptive and programmable MAC schemes, which satisfy changing application demands [18]. Following the flexible MAC designing approach [1] and focusing on WiFi networks, the same research group introduced the idea of wireless MAC processors [19], which is implemented on commodity wireless NICs and supports run-time injection of MAC state machine to program MAC operation without interrupting the MAC service. However, their target scope is limited to IEEE 802.11, and is based on primitives that are offered by selected chipsets. For an extended scope of applications, we have implemented TRUMP [2] for flexible MAC designing and run-time reconfiguration. Although earlier we have shown that the TRUMP philosophy in realizing flexible MAC schemes through wiring of MAC components is feasible on sensor nodes, the complete TRUMP toolchain was not suitable for such resource constrained platforms. In this paper, following the TRUMP philosophy we have extended the toolchain for resource constrained sensor nodes in order to provide both easy MAC design and run-time reconfiguration capability.

III. System Design and Implementation

Based on the component oriented MAC design [1], our toolchain, TRUMP [2] provides run-time MAC protocol realization. It uses a C-like MAC meta-language to simplify the efforts of a designer in prototyping a MAC algorithm. The MAC language is compiled to form MAC schemes for the target platforms. TRUMP enables run-time protocol reconfiguration and adaptation which facilitates MAC performance improvement. The toolchain has been initially implemented on WARP Software Defined Radio platform [20]. As our effort to extend TRUMP into resource constrained platforms, such as sensor nodes, we have designed the overall architecture of our MAC toolchain as shown in Fig. 1. Due to the severe resource constraints, we have offloaded part of the toolchain from the target platform to the more capable PC while providing similar development experience.

As shown in Fig. 1, the MAC protocol description written in user-friendly meta-language is translated to a sensor node interpretable format through a meta-language parser into an array structure. The array is referred to as MetaItem list. The MetaItem list which encompasses the entire MAC description is then passed to sensor nodes through serial communication and executed on the target platform by a MetaItem executor. The MetaItem executor also manages the list of variables used in the protocol and maps MAC components in the component library to the MetaItem for execution at run-time. The system

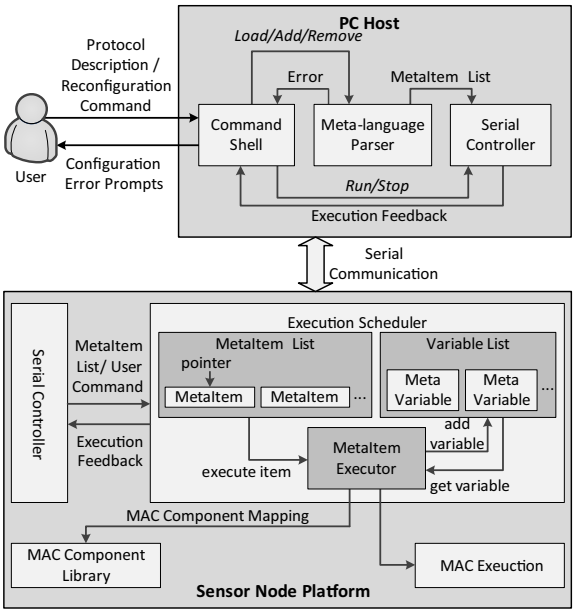

Fig. 1. Architecture of our MAC designing toolchain.

architecture is organized in a way that although a workstation (e.g. PC) is needed for prototyping and debugging of newly designed MAC protocols, sensor nodes operates autonomously for protocol execution and modification. Therefore, run-time protocol reconfiguration of a sensor network can be easily realized by over-the-air programming, i.e. reconfiguration commands can be encapsulated with standard message format to be propagated through the network and the MetaItem executor modifies the MetaItem list by interpreting the message.

Since TinyOS [21] is a commonly used embedded operating system targeting WSNs and supports a wide range of sensor nodes such as TelosB, MICA2 and MICAz from MEMSIC Inc. [22], etc., we have targeted the toolchain for TinyOS. In order to show the portability of our toolchain among various sensor nodes, we have chosen TelosB and MICA2 nodes as our experimental platforms since these two commonly used sensor nodes have different radio chips and microcontrollers. In the rest of the section, we describe the design and implementation details of each part of the toolchain in a bottom-up approach to show how a fully flexible and reconfigurable MAC scheme is realized.

A. MAC Component Library

The toolchain is built on Decomposable MAC Framework [1]. As we have already shown in our work to realize platform independent MAC components in MAC-PD [3], we have extracted a list of generic components including *SendFrame*, *ReceiveFrame*, *RadioPowerControl*, *Timer*, *RandomNumberGenerator*, *CarrierSensing*, *LowPowerListening*, *BinaryExponentialBackoff*, *SendPacket*, *ExpectPacket* and *SendPreamble* which are basic building blocks for both preamble-sampling based and common active period based WSN MAC schemes. Since MAC-PD is targeted for Texas Instruments Inc. CC2420 radio chip [23] based nodes, porting of the needed components for Texas Instruments Inc. CC1000 radio chip [24] (for supporting MICA2 nodes) was required due to differences in the components and interfaces provided by TinyOS. However, the overall porting effort was minimal due to the generic component APIs and all the above identified component interfaces are kept unified and platform independent. The component library resides on sensor nodes.

B. Execution Scheduler

In addition to the MAC functional components, arithmetic, binary and logical expressions are necessary to link the components, declare and process parameters and variables while realizing the complete state machine of a MAC protocol. The Execution Scheduler governs the execution of the MAC protocol by interpreting the expressions to form a correct execution sequence of the MAC components. As shown in Fig. 1, there are three main components in the Execution Scheduler: a MetaItem list which stores the MAC protocol design in the form of a list of elements called MetaItem; a variable list which stores all the variables declared in the MAC design and a MetaItem executor which translates the MetaItem list into executions and manages the variable list accordingly.

1) MetaItem List: Our toolchain uses a linked list structure to describe a reconfigurable MAC protocol. As it is complicated to manage string variables in TinyOS, numerical values are used in the MetaItem structure as shown in Fig. 2. The MetaLabel specifies the type of the MetaItem, for example DEF for a variable definition, FUNC for a function call, etc. IdLabel specifies the identifier of the node, for example the name of the function for a function call while Parameters are a set of attributes which the MetaItem requires.

```
typedef struct MetaItem {
    uint8_t MetaLabel;
    uint8_t IdLabel;
    MetaParam Parameters[5];
}MetaItem;
typedef struct MetaParam{
    uint8_t type;
    uint16_t value;
}MetaParam;
```

Fig. 2. Definition for MetaItem structure.

2) Variable List: There are three types of variables included in an array structure of pre-defined size of 50 elements:

- *System variables*: These are pre-defined by the toolchain to be used globally but they cannot be modified or redefined at run-time.
- *Global variables*: These are defined by the user to be used within the scope of the entire protocol implementation.
- *Local variables*: These are defined by the user within an event implementation and are only valid within a specific event. These variables are removed from the list when the event execution is completed.

3) MetaItem Executor: The MetaItem executor governs the execution sequence of the MAC procedures by managing an execution pointer. The executor first analyzes the MetaLabel in the MetaItem to determine how to interpret the rest of the fields. It adds new variables to the variable list and remove invalid ones. It also translates MetaItems into function calls which are mapped to the MAC component library. The conditional branches such as `IF-ELSE-END_IF` are interpreted and the execution pointer is moved to the appropriate branch according to the conditions specified. The MetaItem executor also points to implementation of the event when it is triggered. An event is realized by defining two specific MetaItems, `EVENT_NAME` which contains the name of the event and `END_EVENT`, which specify the beginning and the end of the event, respectively. Last but not the least, the executor accepts the command for MAC protocol modification at run-time, e.g. to add/remove a MetaItem from the list and adapts the execution accordingly.

C. Meta-language Descriptions and the Parser

In order to provide a rapid MAC prototyping tool without requiring specific knowledge of the target platforms from MAC designers, our toolchain uses a C-like MAC meta-language which simplifies the MAC design process. The language contains logical expressions such as `IF-ELSE-ENDIF`

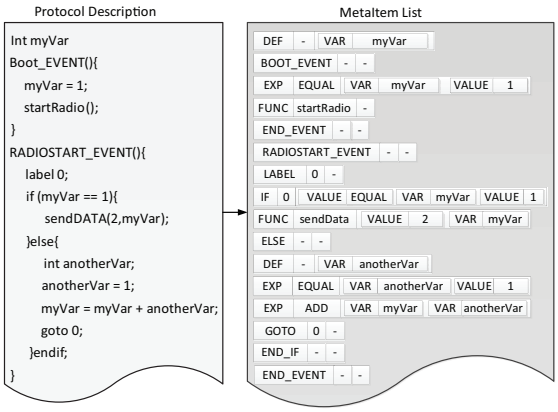

Fig. 3. A mapping of the protocol description in MAC meta-language to the MetaItem list for execution.

and `LABEL-GOTO` conditional clauses. Several modifications have been made to the existing language architecture to suit to the characteristics of sensor nodes. The concept of event is introduced in the language. Since all the callback functions are predefined at compile time, the events associated with function calls are fixed, thus the names of all the events are pre-defined. Each of the event is mapped to a keyword string to be used in the meta-language. `BOOT_EVENT` is a default event that is triggered when the sensor node is booted. When the sensor node is booted, the MetaItem executor will move the execution pointer to `BOOT_EVENT`. Other events are associated with the function calls used in the implementation. For example, in the protocol description in Fig. 3, `startRadio()` function is called in the boot event, therefore `RADIOSTART_EVENT` should accordingly be implemented.

In Fig. 3, the translation procedure from MAC meta-language protocol description to the MetaItem list is illustrated. This is done through a parser implemented in Java. Each line of the protocol is parsed to a MetaItem structure which is fully compatible with the TinyOS structure defined in Fig. 2. MIG (Message Interface Generator) tool is used to provide the structure of the compatible Java MetaItem object. The parser also verifies the meta-language syntax and reports errors, such as variables used before being declared.

D. Command Shell

The command shell provides an interface for the MAC designer to 1) load a new MAC design, 2) execute the loaded MAC design, 3) stop the current MAC execution, 4) add new contents to the currently running MAC protocol, 5) remove contents from the currently running MAC protocols and 6) get helpful information on how to use the command shell. The structure of all the commands are listed in Fig. 4.

IV. Evaluation Results

In this section, we evaluate our toolchain in terms of memory consumption and execution time as compared to their monolithic hand coded counterparts. Since our toolchain introduces extra components to realize run-time reconfigurable

```
>> LOAD [filename]
>> RUN
>> STOP
>> ADD [numLine] [func/exp/def/if-else/
                  label&goto]
>> REMOVE [numLine]
>> HELP [load/add/remove/run/stop/syntax/
        function/variables]
```

Fig. 4. Commands for the shell based control.

solutions, we expect an overhead in terms of memory consumption. In addition, we have also introduced extra layer of abstraction in terms of MAC component implementation which can lead to an execution time overhead. Therefore, these two metrics are the trade-offs we made to enable reconfigurability, portability, and code reusability. All the experiments were carried out on TelosB and MICA2 nodes with CC2420 radio chip and CC1000 radio chip, respectively. Please note that the monolithic implementation of all the MAC protocols in our evaluation is also based on the MAC components we have introduced in Section III-A but without all other parts of the toolchain. A selection of well-known preamble sampling based MAC protocols (i.e. B-MAC [25], MFP-MAC [26] and X-MAC [27]) and common active period MAC protocols (i.e. S-MAC [28] and T-MAC [29]) are implemented for evaluation.

A. Memory Consumption

Since our toolchain is tailored for resource constrained embedded platforms, memory consumption is an important metric. We have measured both RAM and ROM in terms of memory consumption. TABLE I shows the memory footprint results in terms of RAM and ROM consumption for a TelosB node for a list of different types of MAC protocols. *No-protocol* represents a simple procedure which only starts the radio, sends one packet, stops the radio, and repeats this process. We have done similar measurements in our previous related works [2], [3]. Since our toolchain contains both the MAC components similar to the ones identified in MAC-PD [3] and a complete toolset in MAC component wiring and reconfiguration, the results shown here is higher than in our previous work. We can see that the memory consumption for the monolithic implementations is based on the components used and varies among protocols. For the toolchain implementation, since all the components can be wired at

Protocols	***Monolithic***		***Toolchain***	
	ROM	*RAM*	*ROM*	*RAM*
B-MAC	25112	1242	31282	3140
MFP-MAC	25734	1242	31282	3274
X-MAC	25304	1242	31282	3326
S-MAC	24326	1306	31282	4906
T-MAC	22096	1140	31282	4498
No-Protocol	16978	1102	31282	1314

TABLE I
MEMORY FOOTPRINT [BYTE] OF THE IMPLEMENTATION FOR DIFFERENT MAC PROTOCOLS ON TELOSB NODES.

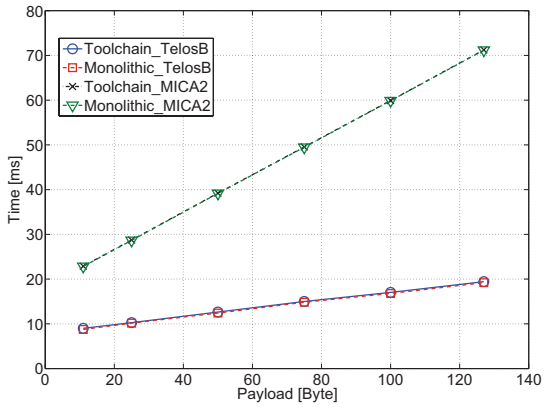

Fig. 5. Duration of sending one packet with different payloads on TelosB and MICA2 nodes with and without our toolchain.

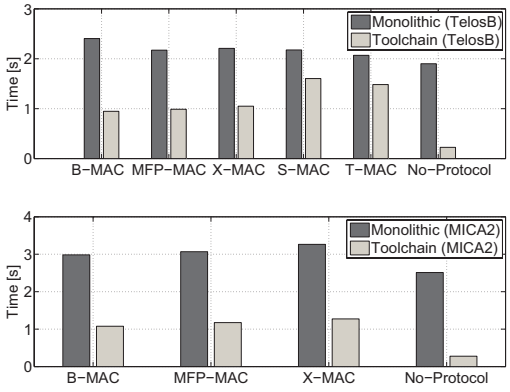

Fig. 6. A comparison of the reconfiguration time on TelosB and MICA2 nodes.

run-time, the ROM consumption remains the same for all protocols. The RAM consumption is dependent on the protocol complexity, i.e. how many MetaItems are required to describe the protocol. A typical 2kB RAM overhead is observed for our implementations. In practise, one can maximize the usage of the memory by declaring a maximum array size for the target platforms to host more complex MAC protocol realizations. It is worth noting that although the toolchain appears to impart a heavy memory consumption overhead, it allows realizations of all MAC protocols and switching among MAC behaviours at run-time without using any extra memory.

B. Execution Time

In order to evaluate how much the execution overhead the toolchain has introduced, we have measured the time taken to send a packet with varying payload with and without the toolchain on sensor nodes as shown in Fig. 5. We have repeated each experiment 10 times to have better statistics and the average is presented. Fig. 5 exhibits a linear relationship between the packet sending duration and the packet size which is as expected. The overhead of using the toolchain is constant and almost negligible for both platforms. The time taken to send a packet of the same size is much longer on MICA2 than on TelosB due to the lower data rate supported by CC1000 as compared to CC2420.

We have also measured the execution time of selected protocols on both TelosB and MICA2 nodes as shown in TABLE II. As shown in Fig. 5, since the absolute overhead is constant, the larger the payload size is, the less significant the overhead becomes. We have chosen a small payload size (11 bytes) for the rest of the expriments to show the worst case scenario for our toolchain. We can see that the overhead percentage added by the toolchain is around 1% in most of the cases on both platforms.

MAC Protocol	*Monolithic*	*Toolchain*	*Overhead*
B-MAC (TelosB)	161.78	163.36	0.97 %
S-MAC (TelosB)	195.84	198.57	1.39 %
No-protocol (MICA2)	26.67	26.88	0.79 %
B-MAC (MICA2)	190.49	191.13	0.33 %

TABLE II
EXECUTION TIME [MILLISECOND] FOR DIFFERENT PROTOCOLS ON TELOSB AND MICA2 NODES.

C. Reconfiguration Overhead

In this section, we evaluate the cost to reconfigure the MAC protocol at run-time. In the traditional monolithic approach, reconfiguration can be achieved only by reprogramming the nodes. The measurements for reconfiguration shown in this section are taken from the user end (a PC which is connected to the target nodes through a serial cable) between the time when a command for reconfiguration is issued and the response received from the target sensor nodes that the protocol has been transformed accordingly. Fig. 6 shows a comparison for the time to load a completely new protocol between the monolithic implementation and reconfigurable approach using toolchain. For the toolchain approach, a new text file is loaded. We can see that in most of the cases 60% of the execution time has been saved. We have also measured the time taken for the protocol to evolve base on reconfiguration commands. In our experiments, we send commands to change the duty cycle of the preamble-sampling protocols and the wake-up interval for the common period protocols. TABLE III shows that the reconfiguration time is independent from the complexity of the protocol and the reconfiguration response time is consistently about 50 ms. Since the time measured involves multiple round-trip communication delay through serial port, the actual reconfiguration time on the target platform is much less.

Protocols	*No. of MetaItems*	*TelosB*	*MICA2*
B-MAC	112	54	45
MFP-MAC	122	48	44
X-MAC	125	41	47
S-MAC	218	50	-
T-MAC	194	54	-
No-Protocol	12	31	47

TABLE III
TIME [MILLISECOND] REQUIRED TO RECONFIGURE A MAC PROTOCOL ON TELOSB AND MICA2 NODES USING OUR TOOLCHAIN.

V. Conclusions

In this paper, we have described a toolchain that enables rapid prototyping of MAC protocols for sensor networks. The toolchain also allows run-time reconfiguration of a MAC scheme in order to cope with the network dynamics as well as to fulfill varying application demands. The toolchain follows a component based design and composes different MAC protocols by binding a set of reusable functional components. While different MAC protocols can be composed from a pool of common MAC components, the exact composition of a particular MAC protocol including its state-machine is unique. Developing protocols using a set of common components lowers the required programming and debugging efforts since all the component implementations are tested. Our toolchain enabled run-time reconfiguration by modifying the binding of MAC components on-the-fly.

We have implemented the toolchain for commercially available and widely used sensor nodes which are supported by TinyOS. We have conducted empirical studies to observe the overhead imparted by our toolchain. Our results show that the protocol execution time overhead is less than 1.4% for the widely used preamble sampling and common active period MAC protocols. Although we have observed a typical 2 kB overhead in RAM for one MAC protocol implementation, our approach allows realizations of a vast list of MAC protocols at run-time using the given limited memory resource. Furthermore, as the demand for large-size low-power RAM increases largely due to the development of smartphones, we believe the availability of RAM should not be a bottleneck in allowing easy and rapid prototyping of MAC protocols. Nevertheless, as a future work, we would aim at further optimizing the memory consumption of the toolchain in order to also accommodate MAC protocols with much higher complexity. The current implementation has been tested to be stable and we are planning to release our toolchain publicly for interested parties. A demonstration video of our toolchain is available online [30].

Acknowledgment

We thank the financial support from Deutsche Forschungsgemeinschaft through UMIC Research Centre, and EU through FP7 funded 2PARMA-project (FP7-ICT-2009-4-248716).

References

[1] J. Ansari, X. Zhang, A. Achtzhen, M. Petrova and P. Mähönen, "Decomposable MAC Framework for Highly Flexible and Adaptable MAC Realizations," in *Proc. of IEEE DySPAN*, Singapore, 2010.

[2] X. Zhang, J. Ansari, G. Yang, and P. Mähönen, "TRUMP: Supporting Efficient Realization of Protocols for Cognitive Radio Networks," in *Proc. of IEEE DySPAN*, Aachen, Germany, 2011.

[3] J. Ansari, X. Zhang, O. Salikeen, and P. Mähönen, "Enabling Flexible Medium Access Design for Wireless Sensor Networks," in *Proc. of IEEE WONS*, Bardoneccia, Italy, 2011.

[4] J. Polakovic, S. Mazare, J.-B. Stefani, and P.-C. David, "Experience with safe dynamic reconfigurations in component-based embedded systems," in *Proc. of ACM CBSE*, 2007, pp. 242–257.

[5] R. Braden, T. Faber and M. Handley, "From Protocol Stack to Protocol Heap: Role-Based Architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 17–22, 2003.

[6] M. A. Yigitel, O. D. Incel, and C. Ersoy, "QoS-aware MAC protocols for wireless sensor networks: A survey," *Computer Networks*, vol. 55, no. 8, pp. 1982–2004, 2011.

[7] J. Ansari, X. Zhang, and P. Mähönen, "A Compiler Assisted Approach for Component Based Reconfigurable MAC Design," in *Proc. of Med-Hoc-Net*, Favignana Island, Sicilly, Italy, 2011.

[8] N. Brouwers, K. Langendoen, and P. Corke, "Darjeeling, A Feature-Rich VM for the Resource Poor," in *Proc. of ACM SenSys*, Berkeley, CA, USA, 2009.

[9] A. Caracas, T. Kramp, M. Baentsch, M. Oestreicher, T. Eirich, and I. Romanov, "Mote Runner: A Multi-language Virtual Machine for Small Embedded Devices," in *Proc. of SENSORCOMM*, USA, 2009.

[10] J. Ellul and K. Martinez, "Demo Abstract: Run-time Compilation of Bytecode in Wireless Sensor Networks," in *Proc. of ACM IPSN*, Stockholm, Sweden, 2010.

[11] J. R. Polastre, "A unifying link abstraction for wireless sensor networks," Ph.D. dissertation, Berkeley, CA, USA, 2005.

[12] K. Rerkrai, J. Riihijärvi, M. Wellens and P. Mähönen, "Unified Link-Layer API Enabling Portable Protocols and Applications for Wireless Sensor Networks," in *Proc. of IEEE ICC*, Glasgow, Scotland, 2007.

[13] K. Klues, G. Xing and C. Lu, "Towards a Unified Radio Power Management Architecture for Wireless Sensor Networks," in *Proc. of International Workshop on Sensor Network Architecture*, Cambridge, Massachusetts, USA, 2007.

[14] G. Xing, M. Sha, G. Hackmann, K. Klues, O. Chipara, and C. Lu, "Towards unified radio power management for wireless sensor networks," *Wirel. Commun. Mob. Comput.*, vol. 9, no. 3, pp. 313–323, 2009.

[15] K. Klues, G. Hackmann, O. Chipara, and C. Lu, "A Component Based Architecture for Power-Efficient Media Access Control in Wireless Sensor Networks," in *Proc. of ACM SenSys*, Sydney, Australia, 2007.

[16] K. Klues, G. Xing, and C. Lu, "Link layer driver architecture for unified radio power management in wireless sensor networks," *ACM Trans. Embed. Comput. Syst.*, vol. 9, no. 4, pp. 1–28, 2010.

[17] O. Salikeen, J. Ansari, X. Zhang, and P. Mähönen, "Demo Abstract: Enabling Flexible MAC Design for Wireless Sensor Networks," in *Proc. of ACM SenSys*, Zurich, Switzerland, 2010.

[18] G. Bianchi and A. Campbell, "A programmable MAC framework for utility-based adaptive quality of service support," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 2, pp. 244–255, 2000.

[19] I. Tinnirello, G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, and F. Gringoli, "Wireless MAC processors: Programming MAC Protocols on Commodity Hardware," in *Proc. of IEEE INFOCOM*, Orlando, Florida, USA, 2012.

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

[21] P. Levis and D. Gay, *TinyOS Programming*. Cambridge University Press, 2009.

[22] "MEMSIC Wrieless Module Products for Wireless Sensor Networks," http://www.memsic.com/products/wireless-sensor-networks/wireless-modules.html [Last visited: 01.10.2012].

[23] "Datasheet: CC2420 - 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver," http://www.ti.com/lit/ds/symlink/cc2420.pdf [Last visited: 01.10.2012].

[24] "Datasheet: CC1000 - Single Chip Very Low Power RF Transceiver," http://www.ti.com/lit/ds/symlink/cc1000.pdf [Last visited: 01.10.2012].

[25] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *Proc. of ACM SenSys*, Baltimore, MD, USA, 2004, pp. 95–107.

[26] A. Bachir, D. Barthel, M. Heusse, and A. Duda, "Micro-Frame Preamble MAC for Multihop Wireless Sensor Networks," in *Proc. of IEEE ICC*, Istanbul,Turkey, 2006.

[27] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks," in *Proc. of ACM SenSys*, Boulder, Colorado, USA, 2006.

[28] W. Ye, J. Heidemann, and D. Estrin, "An Energy-Efficient MAC protocol for Wireless Sensor Networks," in *Proc. of the IEEE INFOCOM*, New York, USA, 2002.

[29] T. van Dam and K. Langendoen, "An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks," in *Proc. of ACM SenSys*, Los Angeles, CA, USA, 2003, pp. 171–180.

[30] "Demonstration Video on the MAC Designing Toolchain for a B-MAC Implementation." http://tinyurl.com/bm7nf5q [Last visited: 15.10.2012].