In: Proceedings of the 18th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'07), Athens, Greece, 3-7 September 2007

A WIRELESS SENSOR MAC PROTOCOL FOR BURSTY DATA TRAFFIC

Luis Bernardo
FCT, Universidade Nova de Lisboa
P-2829-516 Caparica
Portugal
lflb@fct.unl.pt

Rodolfo Oliveira
FCT, Universidade Nova de Lisboa
P-2829-516 Caparica
Portugal
rado@fct.unl.pt

Miguel Pereira
FCT, Universidade Nova de Lisboa
P-2829-516 Caparica
Portugal
miguelpereira.pro@gmail.com

Mário Macedo
Inesc-ID
P-1000-029 Lisboa
Portugal
mmm@fct.unl.pt

Paulo Pinto
FCT, Universidade Nova de Lisboa
P-2829-516 Caparica
Portugal
pfp@fct.unl.pt

ABSTRACT

This paper proposes MH-MAC, a new MAC protocol for wireless sensor networks capable of handling applications that generate infrequent huge peaks of traffic. Existing protocols are not adapted to this kind of applications. Asynchronous protocols are energy efficient for the long inactive periods, but fail to cope with the bandwidth and latency requirements of the traffic peaks when more than two nodes are sending data to a common sink. Synchronous protocols that support contention free slots provide good throughput for handling the load peaks, but consume unnecessary energy maintaining clocks synchronized for very long idle periods. MH-MAC is a multimode hybrid protocol that can be configured by the application to run in asynchronous mode or in synchronous mode, with or without contention, providing the best possible trade-off. MH-MAC is a single-hop MAC, which supports multi-hop applications through a cross-layering API. The paper includes simulation results with the energy consumption, latency and throughput for the operation modes of MH-MAC, showing the asynchronous-synchronous trade-offs and the state transition overhead.

I. INTRODUCTION

Energy efficiency is a dominant concern on the design of the medium access control (MAC) layer protocols for wireless sensor networks (WSNs). Nevertheless MAC protocols must also satisfy the application delay and throughput requirements. Applications that generate infrequent huge peaks of traffic pose a challenging problem for the existing MAC protocols.

Standard WSN MAC protocols are usually designed under the assumptions of periodic traffic, or seldom traffic, but not for applications where both characteristics are needed at different instants. Duty cycling is a common mechanism for achieving energy efficiency. Nodes periodically cycle between an awake state and a sleep state. Protocols designed for seldom traffic, such as B-MAC [1] and X-MAC [2], let nodes run their duty cycles independently. They rely on *low power listening* (LPL), also called preamble sampling, to link together the sender and the receiver asynchronously. Packet sending is preceded by a large preamble, or a sequence of small preambles, larger than the duration of a duty cycle period. Protocols designed for periodic sending, such as S-MAC [3], T-MAC [4], SCP-MAC [5], and Z-MAC [6], require the additional clock synchronization overhead. Nodes run synchronized duty cycle periods. S-MAC [3] is a periodic synchronous protocol, which runs a CSMA (Carrier Sense Multiple Access) MAC contention resolution protocol during the fixed duration of the awake state. T-MAC [4] improves S-MAC by adapting the awake state duration to the load. If the radio is inactive for more than a threshold time the node goes asleep before the end of the normal awake duration time. SCP-MAC [5] introduces the scheduled channel polling technique to achieve awake duty cycle values as low as 0.01%. During a very short awake time nodes scan the medium for energy. If energy is detected, nodes stay awake and wait for a packet reception. This mechanism requires very precise synchronization between the sender and the receiver, due to the small durations proposed for the awake state and for the sender's awake signals (packets). A common problem for CSMA contention based protocols are collisions with nodes two hop away, called the *hidden terminal* problem. A common solution to the problem is the RTS/CTS exchange. However, this solution can incur in high bandwidth overhead [1]. Z-MAC [6] improves CSMA using a TDMA contention free mode when the traffic increases. It introduces the overhead of creating and maintaining a global slot schedule. Nonetheless, WSN applications often create a sink tree [7] where the trunk links demand much more throughput than the leaf links. Z-MAC fails to cope with such applications because it divides TDMA slots evenly amongst neighbor nodes. Bursty traffic presents a challenge to the synchronous WSN MAC protocols, due to the high synchronization overhead paid during the idle periods, when no packets are flowing. On the other hand, asynchronous protocols delay packet sending and limit the maximum throughput.

This paper proposes MH-MAC, a multimode hybrid MAC protocol that is capable of running in asynchronous mode and synchronous mode. MH-MAC allows applications to operate in the asynchronous mode for most of the time, and change to contention-free synchronous mode during the data traffic peaks, optimizing the overall performance.

In the following, Section 2 presents the MH-MAC protocol. MH-MAC application programming interface is described in Section 3. Section 4 evaluates the protocol performance using TOSSIM [8] simulations. Section 5 provides a discussion of future work and our conclusions.

II. MH-MAC DESIGN

Multimode hybrid MAC protocol (MH-MAC) is designed to support cross-layering applications for packetizing radios, like the Chipcon CC2420. MH-MAC can be in one of three states: asynchronous state; synchronous state; or the full-on state, where the node does not sleep. By default MH-MAC state is asynchronous, but applications can change it to full-on, or synchronous. In the full on state, data packets are preceded by an RTS/CTS exchange, and are acknowledged. In the synchronous state, temporarily contention-free slots can be reserved for the communication with neighbor nodes,

trading off energy for throughput and delay. The following subsections present the MH-MAC operation modes associated with the two duty cycling states, and the state transition protocol.

A. Asynchronous Mode

The MH-MAC asynchronous mode runs a LPL algorithm similar to the X-MAC protocol [2]. Senders send a sequence of short preambles during at most twice the time of the duty cycle period, before sending the data packet. This assures that the receiver is awake when the data packet is sent. The preambles contain the destination address and are separated by pauses. This allows MH-MAC unicast receivers to send early preamble acknowledgments as soon as they awake and receive a preamble, shortening the unicast preamble durations (Fig. 1). However, for broadcast, the full preamble is required. MH-MAC improves X-MAC overhearing protection using an additional field in the preamble: the missing preamble time to the data packet transmission. Broadcast receivers (Fig. 2) use this field to schedule a radio sleep until the beginning of the data packet transmission. Unicast senders use this field to schedule a precise sleep period when they are waiting to send a packet and receive preambles destined to someone else.

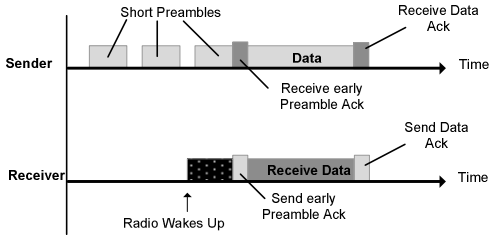

Figure 1: Unicast asynchronous transmission.

MH-MAC handles preamble collisions using the SHUT-UP packet. Receivers send a SHUT-UP packet to the recent senders with a probability *p* when it hears more than one preamble. The SHUT-UP packet includes the active sender address and its missing preamble time, allowing the other senders to back off and sleep until the end of the packet transmission. Inter-preamble time includes a jitter that improves the probability of not having preamble collisions between concurrent senders.

Only unicast data packets are acknowledged. After a data packet reception, the receiver stays awake for a short period, waiting for possible new packets before returning to sleep.

B. Synchronous Mode

The MH-MAC synchronous mode was designed to optimize data collection from a distributed set of sensor nodes into a single sink, over a sink tree. Therefore, it provides basic flooding, neighbor detection, and slot conflict resolution functionalities. MH-MAC organizes the duty cycle period into a sequence of fixed length slots (100 ms). Each slot is capable of carrying an average of 11.87 data packets with 112 bytes on 802.15.4 radios. Nodes run synchronized duty cycle periods. Each node has one or more public slots, and zero or more dedicated slots to communicate with specific neighbors.

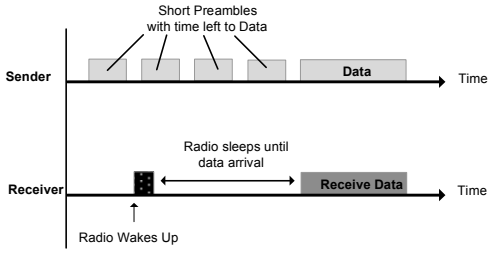

Figure 2: Broadcast asynchronous transmission.

On public slots nodes run a contention-based protocol similar to T-MAC [4]. Senders run a backoff timer and scan the network before sending a data packet. Unicast packets are acknowledged and are preceded by a RTS/CTS exchange when their length is above a threshold value. If no energy is detected in the channel in a public slot for 25 ms, the receiver node goes into sleep. The public slot supports premium and regular traffic. The first 15 ms are reserved for MAC signaling and premium packets (usually application signaling). The remaining 85 ms can be used by all kinds of traffic. Dedicated Slots are reserved for unicast communications between two nodes. Senders run a short random backoff timer before scanning the networks' energy and sending the data packets. Data sent through dedicated slots is also acknowledged, but no RTS/CTS packets are sent.

In order to maintain synchronization, nodes send sparse SYNC packets on their public slots, defining the beginning of the duty cycle period. The SYNC packet contains the local address, the local slot assignment plan, and a hop counter. Initial slot assignment is done radially, from the sink to the farthest located nodes, taking into account the slots occupied by neighbor nodes. Each node keeps track of its public slot, its dedicated slots, and the slots occupied by other neighbor nodes. A SYNC packet has a maximum validity of 180 seconds, and its information is discarded after that time. Nodes are asleep during empty slots or slots occupied by neighbors. If a node has more than one public slot, it runs the contention-based access algorithm present above for each slot, possibly with distinct groups of nodes.

Nodes in asynchronous or full-on modes can exist in the neighborhood of nodes in synchronous mode. In these cases, asynchronous and full-on nodes must maintain a table with the slots of each synchronous neighbor node, and start the packets transmission on a public slot. When the destination node state is unknown, synchronous nodes must send a preamble preceding the data packets. If the awake periods are aligned, the preamble overhead is minimized by the immediate reception of an early preamble acknowledge. After the state transition period, collisions in dedicated slots can be avoided on static WSN networks only if all nodes are set to the synchronous state.

Fig. 4 illustrates a possible slot allocation schedule for the sink tree shown in Fig. 3. An eleven slot duty-cycle period is used and dedicated slots were allocated for all connections. All nodes are within two hops of the sink and share the same public slot (0). Fig. 4 presents the slots occupied by neighbors in grey, and the local dedicated slots in black. Node A uses the slots 3, 6, and 10 respectively to communicate with nodes B, D and C. Node A has slots 5, 8, and 9 occupied by

neighbor nodes C and D. Dedicated slots in Fig. 4 are organized in a staggered wakeup schedule that minimizes the source to sink delay [9]. Although MH-MAC only manages synchronous connections at a local level, its API (application programming interface) allows applications to specify the slot assignment. By default, MH-MAC distributes slots randomly. In order to minimize the packet propagation delay from a branch node to the sink node, the application must select the nearest free slot before the sink's slot. In Fig. 4, C's would search backwards for a free slot starting on A's slot (10).

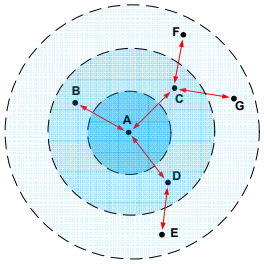

Figure 3: A synchronous WSN topology example.

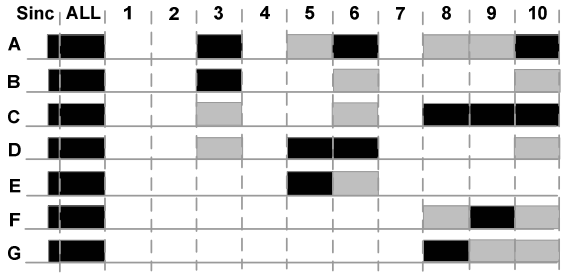

Figure 4: Example slot assignment for Fig.3's WSN.

C. *Mode Swapping*

Applications can modify the MH-MAC state using the API presented below. State changes are notified to the neighbor nodes using *Hello* packets. These packets have an MH-MAC state field and a *slot reserve bit* (SRBit). *Hello* packets are broadcasted using the asynchronous mode algorithm to allow all neighbor nodes to detect the state changes.

The most demanding state change occurs from the asynchronous state to the synchronous state. Fig. 5 illustrates the packets exchanged during this transition. Node A broadcasts the *Hello* packet preceded by a sequence of preambles, signaling the synchronous state. After receiving this packet, nodes B and C send a *SynchronizeReq* event to their applications that includes the SRBit and the sender address. The application can decide to accept a slot request or to ignore it, but MH-MAC layer always store the A's public slot schedule. The receiving nodes start a random backoff timer, sense the network for other transmissions, and finally transmit a *Request* packet. If the node does not receive an *OK* packet, it restarts the backoff timer and repeats the procedure, until a maximum time of 1100ms after the reception of the *Hello* packet.

The *Request* packet also carries a SRBit. The *Request*/*OK* exchange is used to assign a dedicated slot when the SRBit in the *Request* packet is set, or to free it otherwise. When the synchronous node (A in Fig. 5) sends a *Hello* packet with the SRBit set, it requests every neighbor to ask for dedicated slots. The synchronization node assigns the dedicated slots for each *Request* packet received. The synchronization process ends at the beginning of the next duty cycle period (if it is already defined by a neighbor), and after waiting for a minimum of 1100ms. At that instant, MH-MAC sends a *SynchronizeDone* event to the sender's application with the list of neighbors and the dedicated slots assigned. It also broadcasts a SYNC packet, defining the instant when the neighbors can also start their synchronization process. The application on each neighbor can start its synchronization independently. The duration of the state transition from the asynchronous state to the synchronous state on each node is equal to the preamble duration (twice the duty cycle period), plus a varying time between 1100 ms and 1100 ms plus the duty cycle time.

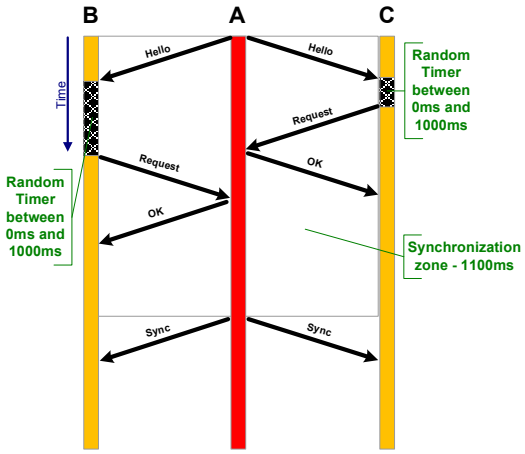

Figure 5: Asynchronous to synchronous state change.

Nodes must change their state to synchronous, starting from the sink node, to synchronize the duty-cycle periods within the sink tree. In order to avoid packet collisions, only one node can run the state change protocol within one hop radio range at each instant. Interferences can occur at two hop distant nodes or more [10]. However, nodes use a RSSI interference detection method to avoid collisions. Even if the initial synchronization fails, nodes can still change their state when a SYNC packet is latter received. If the radio interference effects are ignored, the maximum time to change all nodes on a WSN to the synchronous state can be given by (1), where v_{max} represents the maximum number of neighbors, r represents the depth of the sink tree, and T the duty cycle period. The effective time can be lower because some nodes have less than v_{max} neighbors, and we are considering the maximum possible value for the individual synchronization time. In order to have a faster transition to the synchronous state, the cycle time period could be shorter, resulting in less dedicated slots available and a more awake time. For the eleven slot duty cycle period presented in Fig. 4, T is equal to 1100ms. The time to set the seven nodes presented in Fig. 3 to synchronous mode would be 26.4 seconds.

$$TotalSyncTime \le (2 + r(v_{\max} - 1)) \times (3T + 1100)[ms] \quad (1)$$

Dedicated slots can also be canceled, created or reassigned after the initial synchronization setup phase, running the *Request*/*OK* packet exchange. Due to the limited number of slots available some neighbors may not have dedicated slots assigned to them. These neighbors can use the public slots for data transfer. MH-MAC allows applications to reassign the slots. MH-MAC API allows the application to get the currently assigned slots and to reassign them, accordingly to the active application sink tree.

III. Application Programming Interface

In order to support the cross-layering interaction between MH-MAC and the application and transport layers, the API defines a set of commands (com.) and events. Table 1 presents a subset of the API, organized in pairs of associated commands and events. The commands interface is implemented by MH-MAC and the event interface is implemented by the applications.

Table 1: Application Programming Interface

Type	Command
com.	*Asynchronize*
event	*AsynchronizeDone*
com.	*FullOn*
event	*FullOnDone*
com.	*Synchronize (addr, dedic, [slot])*
event	*SynchronizeDone(neig[],dedic[])*
event	*SynchronizeReq(addr,dedic)*
com.	*ContinueReq(dedic)*
com.	*StopSynchronize(addr)*
event	*SynchronizeKilled(addr)*

The *Asynchronize*, *FullOn*, and *Synchronize* commands are used to modify the MH-MAC state. The associated events *AsynchronizeDone*, *FullOnDone*, and *SynchronizeDone* are generated when the state change protocol ends. The *Synchronize* command starts the synchronization protocol presented in the previous section when *addr* is the broadcast address. The SRBit is set to true when *dedic* is not zero. When *addr* is an unicast address, it starts a *Request*/*OK* packet exchange to assign *dedic* dedicated slots, optionally defining the requested slots. The *SynchronizeDone* event returns the neighbor list and the slots assigned to each neighbor. The *SynchronizeReq* event is received when the HELLO packet is received, and the application uses the *dedic* parameter of the *ContinueReq* command to accept or ignore the request. The application can use the *StopSynchronize* to free a dedicated slot. It can be reassigned after receiving the *SynchronizedKilled* event. Additional commands are available to control the duty cycle period duration, to get the current MH-MAC state, and to send premium messages.

IV. Performance Evaluation

The MH-MAC prototype was implemented in TinyOS 2.0 [11] and was tested on Xbow Telos B motes [12]. However, due to the small number of motes available for this project, the performance evaluation was done using the TOSSIM simulator [8]. The current TOSSIM version does not support the CC2420 radio stack used by the LPL library. Therefore, we emulate the CC2420 radio stack and modified TOSSIM interface implementations to resolve the synchronization problems that occurred when the radio interface is turned off. Additionally, meters were placed on the MAC code to measure the number of milliseconds used for data transmissions, for data receptions, and the time spent in active and radio sleep states. Using the current consumption specifications shown in Table 2, we were able to estimate the total current consumption for the three states of MH-MAC. Following [2][5], we considered that in idle or receiving state the mote has the consumption of operation MCU+Radio RX, in radio sleep has the consumption of operation MCU Active, and during packet transmissions has the consumption of operation MCU+Radio TX.

Table 2: Xbow Telos B current consumption [12]

Operation	Current
Mote Standby (RTC on)	5,1 μ A
MCU Idle (DCO on)	54,5 μ A
MCU Active	1,8 mA
MCU + Radio RX	21.8 mA
MCU + Radio TX (0dBm)	19,5 mA

We analyzed a single-hop scenario where several nodes send packets to a single sink, for the three MH-MAC modes. A duty cycle period with eleven slots is used on the synchronous mode, supporting one public slot plus ten dedicated slots. In the synchronous state nodes generate SYNC packets every 60 seconds. Nodes generate 100 bytes data packets spaced with an average inter packet time (IPT). The interval between packet transmissions is a random variable with a uniform distribution in the interval [0.5xIPT, 1.5xIPT]. The load is uniformly distributed over the nodes.

Fig. 6 shows how the current consumption depends on the IPT value, with four active senders. Results show that when the interval between data packets is large, asynchronous mode (X-MAC) optimizes energy savings because it maximizes the time the nodes are asleep. However, when IPT is small, its current consumption increases significantly due to the preamble overhead. Synchronous mode presents the best energy efficiency for high data rate conditions, where IPT is very short.

Fig. 7 shows the maximum throughput measured when a varying number of nodes send a burst of 20 packets to a sink node at the MH-MAC mode's maximum speed. Results show that the asynchronous mode (X-MAC) is very efficient when one or two neighbor nodes compete, as reported in [1][2]. However, they also show that the asynchronous mode's throughput, and therefore B-MAC and X-MAC's throughput, collapse when three or more senders compete for the access to a single sink. For more then four senders, the synchronous mode outperforms all other modes, maintaining a controlled current consumption.

Fig. 8 shows how the latency evolves with the number of nodes, for an IPT of 1 second. It clearly shows that asynchronous mode is only effective for up to two active senders in the simulated conditions. For more than two active senders, the network latency increases fast with the number of

senders. Full-on mode is clearly the mode that minimizes the network latency, at the cost of also maximizing the current consumption. Therefore, synchronous mode presents the best trade-off for a tree shaped network, with more than two active neighbors sending data to a single sink node.

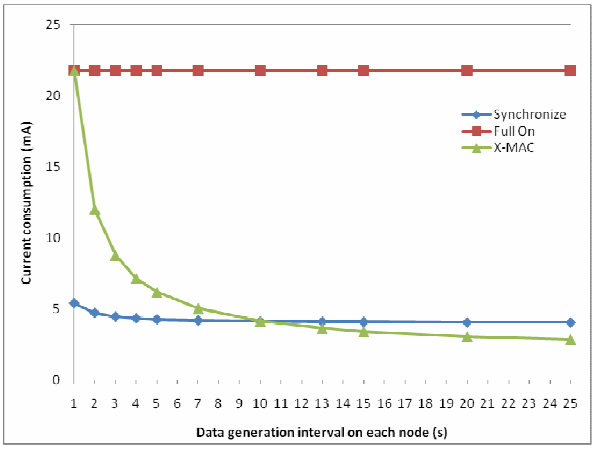

Figure 6: Current consumption with 4 nodes and one sink.

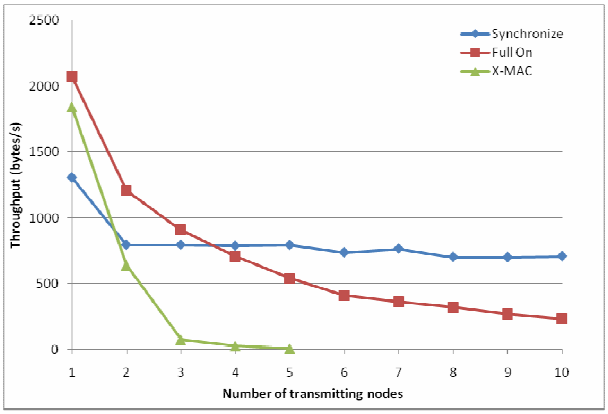

Figure 7: Throughput for a burst of 20 packets per node.

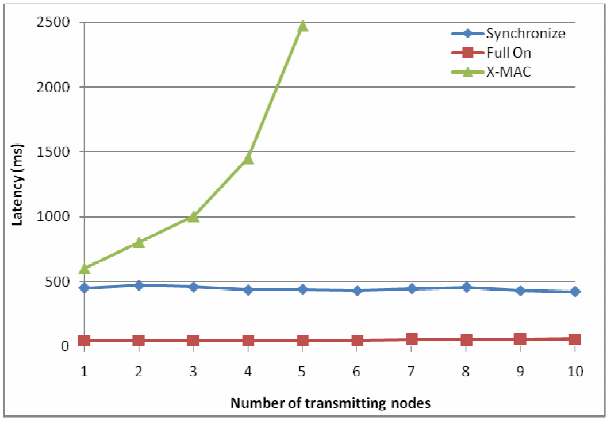

Figure 8: Latency for a burst of 20 packets per node.

V. Conclusions and Further Work

In this paper we propose MH-MAC, a new multimode MAC protocol designed specifically for cross-layering applications. Simulation results show that each mode has a different scenario where it is advantageous: asynchronous for low power sporadic communication; full-on for very low delay; synchronous for handling data bursts. The synchronous mode setup delay can be amortized if large bulks of data are transferred. MH-MAC is being used for the development of asynchronous sensor monitoring and alarm applications in scattered WSNs. Mobile vehicles drive around the WSNs from time to time, collecting all the data stored since the last visit. MH-MAC improves the energy efficiency.

Future work in MH-MAC includes the improved handling of radio interference (inter-slot and on synchronization deployment), and the reduction of the asynchronous to synchronous transition overhead. We also envision improved WSN mobility support.

VI. Acknowledgements

We thank David Moss for the support on the development of MH-MAC. This work was partially supported by the *Fundação para a Ciência e Tecnologia* under the project SIGAPANO POSC/EIA/62199/2004.

References

[1] J. Polastre, J. Hill, D. Culler, "Versatile low power media access for wireless sensor networks," in: *Proceedings of the 2nd ACM Int. Conf. on Embedded Networked Sensor Systems (SenSys)*, 2004, pp. 95–107.

[2] M. Buettner, G. Yee, E. Anderson, R. Han, "X-MAC: A Short Preamble MAC Protocol for Duty-Cycled Wireless Sensor Networks," in: *Proceeding of the 4th ACM International Conference on Embedded Sensor Systems (SenSys)*, 2006, pp. 307-320.

[3] W. Ye, J. Heidemann, D. Estrin, "An energy-efficient MAC protocol for wireless sensor networks," in: *Proceedings of the Joint Conference of the IEEE Computer and Communications Societies (InfoCom)*, Vol. 3, 2002, pp. 214–226.

[4] JT. van Dam, K. Langendoen, "An adaptive energy-efficient MAC protocol for wireless sensor networks," in: *Proc. of the ACM Int. Conf. on Embedded Networked Sensor Systems (SenSys)*, 2003, pp. 171–180.

[5] Wei Ye, Fabio Silva, and John Heidemann, "Ultra-Low Duty Cycle MAC with Scheduled Channel Polling," in: *Proc. of the ACM Int. Conf. on Embedded Networked Sensor Systems (SenSys)*, 2006, pp. 321-334.

[6] II. Rhee, A. Warrier, M. Aia, J. Min, "Z-MAC: A hybrid MAC for wireless sensor networks," in: *Proceedings of the ACM Int. Conference on Embedded Networked Sensor Systems (SenSys)*, 2005, pp. 90–101.

[7] K. Akkaya, and M. Younis, "A survey on routing protocols for wireless sensor networks," *Ad Hoc Networks*, vol. 3, no. 3, pp. 325-349, May 2005.

[8] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications," in: *Proc. of the ACM Int. Conf. on Embedded Networked Sensor Systems (SenSys)*, 2003, pp. 126-137.

[9] G. Lu, B. Krishnamachari, and C. Raghavendra, "An Adaptive Energy-Efficient and Low-Latency MAC for Data Gathering in Wireless Sensor Networks," in: *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2005, pp. 224- 231.

[10] J. Grönkvist, A. Hansson, "Comparison between Graph-Based and Interference-Based STDMA Scheduling", in: *Proceeding of ACM Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc)*, 2001, pp. 255-258.

[11] TinyOS 2.0 Documentation. http://www.tinyos.net/tinyos-2.x/doc/.

[12] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling Ultra-Low Power Wireless Research," in *Proceeding of Int. Symposium on Information Processing in Sensor Networks (IPSN)*, 2005, pp. 364- 369.